VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Server pro podporu výuky teoretické informatiky

Supporting Server for Theoretical Computer Science Teaching

2016 Bc. Michal Kubinec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Michal Kubinec**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Server pro podporu výuky teoretické informatiky
Supporting Server for Theoretical Computer Science Teaching

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je vytvořit server, na kterém si studenti budou moci spustit výpočet vybraných algoritmů z oblasti teoretické informatiky na jimi zadaných i ukázkových vstupech a na zobrazeném průběhu výpočtu lépe pochopit princip fungování těchto algoritmů. Tato práce se konkrétně zaměří na převody mezi NP-úplnými problémy.

1. Naprogramujte alespoň 5 algoritmů převodu mezi NP úplnými problémy (z různých oblastí).
2. Cílem není nejefektivnější implementace algoritmů, ale taková, která umožní zobrazovat postupně jednotlivé kroky tak, aby uživatel mohl pochopit princip jejich fungování.
3. Vytvořte výukový server, kde bude možné zadat vstupní data pro naprogramované algoritmy a pro tato data si nechat zobrazit postup výpočtu.
4. Vytvořte i ukázkové vstupy pro jednotlivé algoritmy, aby postup výpočtu bylo možné zobrazit i bez zadávání vstupu uživatelem.
5. Návrh přizpůsobte snadnému přidání dalších převodů mezi vámi uvažovanými problémy i mezi jinými.

Seznam doporučené odborné literatury:

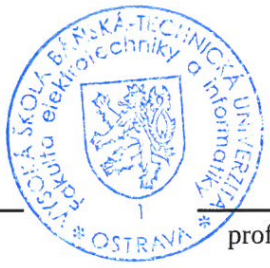
[1] P. Jančar: Teoretická informatika (výukový text), FEI VŠB - TUO, 2007.

Další literatura dle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 01.09.2015
Datum odevzdání: 29.04.2016



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 20. dubna 2016 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 20. dubna 2016 .....

Rád bych poděkoval vedoucímu své diplomové práce panu Ing. Martinu Kotovi, Ph.D. za odbornou pomoc a konzultaci při jejím vytváření.

Abstrakt

Tato diplomová práce popisuje server pro podporu výuky teoretické informatiky, konkrétně zaměřený na převody mezi NP-úplnými problémy. Server, kromě toho že obsahuje potřebnou teorii k dané oblasti, umožňuje uživatelům zobrazit průběh vybraných převodů mezi NP-úplnými problémy a na zobrazeném průběhu výpočtu lépe pochopit princip fungování těchto algoritmů. Samotný text této práce se pak věnuje popisu nezbytné teorie, analýze, návrhu a zejména popisu implementace řešení jako takové, včetně použitých technologií.

Klíčová slova: NP-úplné problémy, polynomiální převoditelnost, webová aplikace

Abstract

This diploma thesis describes supporting server for theoretical computer science teaching, specifically focusing on reductions between NP-complete problems. Server, except that it contains necessary theory of the topic, allows users to view progress of selected reductions between NP-complete problems and on displayed calculation to better understand principles of working of these algorithms. The text of this work deals with description of essential theories, analysis, design and especially description of implementation of the solution, including technologies used.

Key Words: NP-complete problems, polynomial-time reduction, web application

Obsah

Seznam použitých zkratek a symbolů

.NET	–	Software Framework developed by Microsoft
API	–	Application Programming Interface
ASP	–	Active Server Pages
C#	–	C-Sharp, programovací jazyk
CG	–	Coloring Graph
CSS	–	Cascading Style Sheets
GUI	–	Graphical User Interface
HC	–	Hamiltonovský Cyklus
HK	–	Hamiltonovská Kružnice
HTML	–	Hyper Text Markup Language
ILP	–	Integer Linear Programming
IS	–	Indipendent Set
JSON	–	JavaScript Object Notation
MIT	–	Massachusetts Institute of Technology
REST	–	Representational State Transfer
SAT	–	Satisfiability
SVG	–	Scalable Vector Graphics
TFS	–	Team Foundation Server
TSP	–	Travelling Salesman Problem
URI	–	Uniform Resource Identifier

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Problematika NP-úplných problémů je oblast teoretické informatiky, která je velmi důležitá např. v moderní kryptografii. Zde totiž musíme být schopni rychle ověřovat správnost řešení, ale naopak jeho nalezení musí trvat dlouho. Díky převodům mezi NP-úplnými problémy jsme schopni najít řešení všech problémů patřících do této třídy tím, že najdeme řešení jednoho takového problému. Proto mohou být pro nás tyto převody tak důležité a užitečné.

Cílem této diplomové práce je navrhnout a naimplementovat server pro podporu výuky této oblasti. Kromě nastudování nezbytné teorie by si na něm studenti měli moci spustit výpočet vybraných převodů mezi NP-úplnými problémy a na zobrazeném průběhu výpočtu lépe pochopit princip fungování těchto algoritmů. Průběhy převodů by měly být zobrazeny korektně jak pro ukázkové vstupy, tak pro vstupy zadané uživatelem. Cílem není co nejefektivnější implementace těchto algoritmů, ale taková, která umožní pro jakýkoliv možný vstup zobrazovat postupně jednotlivé kroky tak, aby uživatel mohl pochopit principy jejich fungování. Návrh by pak měl být přizpůsoben snadnému přidání dalších převodů jak mezi naimplementovanými problémy, tak mezi jinými.

Samotná diplomová práce je rozdělena do dvou hlavních částí. První část (kapitola 2) obsahuje základní teorii dané problematiky, popis naimplementovaných NP-úplných problémů a převodů mezi nimi, včetně zdůvodnění korektnosti těchto převodů. Ve druhé části (kapitoly 3 4 5) se nachází analýza, návrh uživatelského rozhraní a zejména popis implementace samotného řešení, včetně použitých technologií.

Stěžejní částí je kapitola 5. Z počátku je zde popsán postup a vývoj celého projektu, použité technologie, frameworky a knihovny. Další podkapitoly se věnují všem možnostem zadávání vstupů a popisům jednotlivých simulací převodů. Jsou zde také popsány vybrané algoritmy a konfigurační možnosti aplikace. Předposlední kapitola 6 pak ještě popisuje nasazení výsledné aplikace na webový server.

2 Teorie NP-úplných problémů

Tato kapitola obsahuje teorii, popis oblasti a konkrétní popis NP-úplných problémů a převodů mezi nimi, které byly implementovány v rámci této diplomové práce.

2.1 Základní teorie

V této podkapitole jsou vysvětleny základní pojmy a teorie dané problematiky.

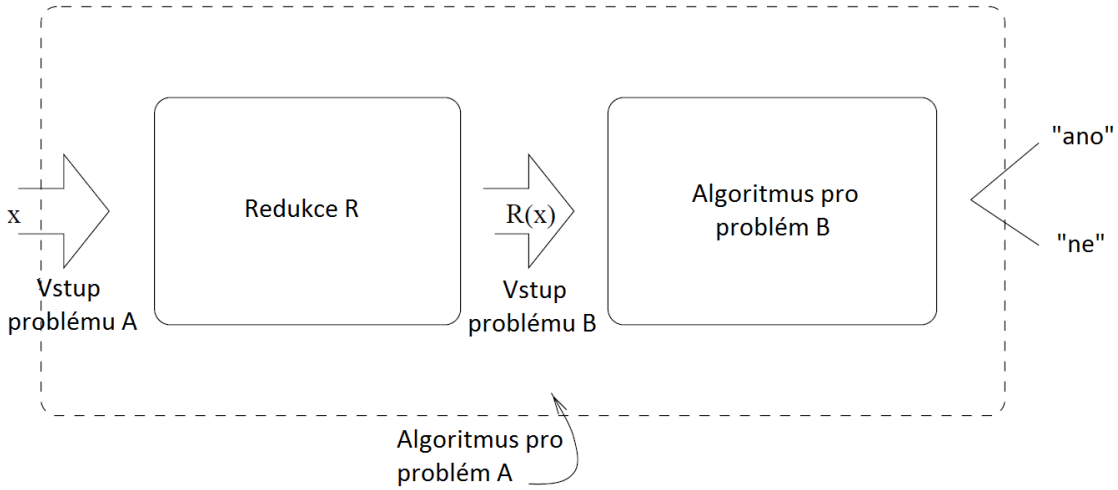
Třída P Úloha (algoritmus) je řešitelná v polynomiálním čase, pokud její časová složitost je shora ohraničená polynomem. Třída úloh s polynomiální složitostí se značí P a tyto úlohy se považují za řešitelné v přiměřeném čase.

Třída NP Úlohu nazveme nedeterministicky polynomiální, jestliže existuje nedeterministický algoritmus, který ji řeší v polynomiálním čase. Tato třída úloh se značí NP. U těchto úloh nám hrozí, že počet možností, které se musí projít na pro ně nejhorších instancích, je exponenciální (2^n) vzhledem k velikosti dané instance.

Polynomiální převoditelnost Při řešení nějaké algoritmické úlohy můžeme přijít na přímé řešení využívající základní techniky. Někdy se může i stát, že v problému rozpoznáme jiný problém, který můžeme převést například na třídění čísel anebo jej umíme popsat nějakým vhodným grafem apod. Proto se ve třídě NP problémy často převádí, jelikož přímá řešení jsou zde vzácná.

Polynomiální převoditelnost je definována takto:
Pokud máme 2 problémy A, B, jejichž výstupem je ANO/NE, tak problém A je polynomiálně převeditelný na problém B, jestliže existuje (převádějící/redukční) polynomiální algoritmus R, který pro jakýkoliv vstup x problému A sestrojí vstup problému B, označme jej $R(x)$ a zároveň platí, že odpověď na otázku problému A pro vstup x je ANO právě tehdy, když odpověď na otázku problému B pro vstup $R(x)$ je taktéž ANO. [1]

NP-úplné problémy Jsou nedeterministicky polynomiální úlohy ze třídy NP, na které jsou polynomiálně převoditelné všechny ostatní problémy z NP. Cookova věta [2] nám říká, že existuje alespoň jeden takový problém, tedy že NPC třída je neprázdná - samotná definice NP-úplného problému totiž nezaručuje, že takový problém vůbec existuje. Cookovu větu tedy lze využít k dokazování úplnosti převodem. Víme totiž, že všechny NP-úplné problémy (kterých mimochodem bylo dosud popsáno přes 2000 různých a nekonečně mnoho jejich variant) jsou mezi sebou



Obrázek 1: Polynomiální převoditelnost

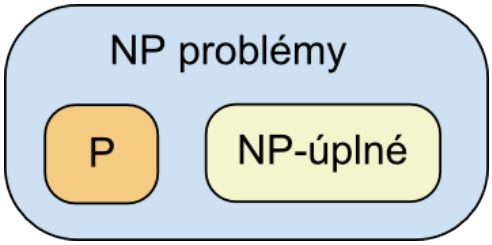
převoditelné (tam i zpět) v polynomiálním čase, neboli jsou na sebe redukovatelné. K dokázání toho že problém je NP-úplný, nám potom stačí jen následující dva kroky: [3]

- Dokázat, že tento problém je v NP (viz odstavec Třída NP)
- Převést zadání libovolného NP-úplného problému na zadání dokazovaného problému tak, že existence polynomiálního algoritmu pro dokazovaný problém by znamenala existenci polynomiálního algoritmu pro řešení onoho použitého známého NP-úplného problému (viz odstavec Polynomiální převoditelnost)

P-NP problém Pokud by byl nalezen polynomiální deterministický algoritmus pro nějakou NP-úplnou úlohu, znamenalo by to, že všechny nedeterministicky polynomiální problémy jsou řešitelné v polynomiálním čase, tedy že třída NP se „zhroutí“ do třídy P (NP = P). Naopak, pokud by se prokázalo, že pro kterýkoliv NP-úplný problém neexistuje polynomiální algoritmus, znamenalo by to, že neexistuje polynomiální algoritmus pro žádný NP-úplný problém. Otázka, zda nějaký takový algoritmus existuje, zatím nebyla rozhodnuta, předpokládá se však, že NP $\neq$ P (je však zřejmé, že P $\subseteq$ NP), viz obrázek 2. Zjednodušeně řečeno jde o otázku, zda každý problém, u kterého dokáže počítač rychle ověřit správnost nabídnutého řešení, dokáže počítač také sám rychle vyřešit. Vztah mezi třídami složitosti P a NP je tudíž jedním z tzv. problémů tisíciletí, za jehož rozhodnutí je vypsána odměna 1 milion dolarů. [4]

2.2 Příklady NP-úplných problémů

Tato podkapitola obsahuje konkrétní popis všech NP-úplných problémů, které byly součástí implementace v rámci této diplomové práce.



Obrázek 2: P-NP problém

2.2.1 Problém 3-SAT

Název: Problém splnitelnosti booleovských formulí s omezením na 3 literály v klauzuli

Vstup: Booleovská formule v konjunktivní normální formě, kde v každé klauzuli jsou právě 3 literály. V konjunktivní normální formě je taková formule, která je ve tvaru konjunkcí klauzulí. Klauzule je potom definována jako disjunkce literálů, kde literál je buď booleovská proměnná nebo negace booleovské proměnné. Každá samostatná konjunkce literálů a taktéž každá samostatná disjunkce literálů je již z tohoto principu v konjunktivní normální formě, jelikož je můžeme považovat za konjunkci klauzulí s jedním literálem, respektive za konjunkci jedné klauzule.

Výstup: Odpověď na otázku, zda je vstupní formule splnitelná. Tzn., že se ptáme, zda existuje pravdivostní ohodnocení proměnných, při kterém je formule pravdivá.

Příklad formule v konjunktivní normální formě: $(A \wedge B) \wedge \neg C \wedge (A \wedge \neg B \wedge C)$

Příklad 3-SAT formule: $(A \wedge \neg B \wedge C) \wedge (\neg A \wedge B \wedge C) \wedge (A \wedge \neg B \wedge \neg C)$

Výše zmíněná 3-SAT formule je splnitelná, jelikož je pravdivá např. pro ohodnocení $[A = 1, B = 1, C = 0]$.

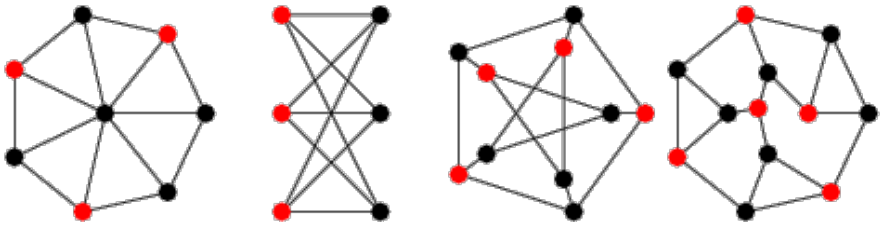
2.2.2 Problém IS

Název: Problém nezávislé množiny

Vstup: Neorientovaný graf G (o n vrcholech), číslo k ($k \leq n$)

Výstup: Odpověď na otázku, zda v grafu G existuje nezávislá množina velikosti k. Jinými slovy se ptáme, zda existuje v grafu G množina k vrcholů, z nichž žádné dva nejsou spojeny hranou.

Příklad: Na obrázku 3 můžeme vidět ve čtyřech různých grafech nezávislé množiny skládající se z červených vrcholů.



Obrázek 3: Příklady nezávislých množin [5]

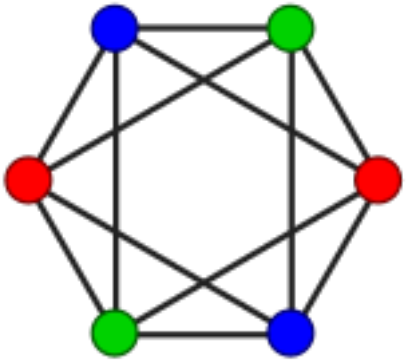
2.2.3 Problém 3-CG

Název: Problém vrcholového obarvení grafu třemi barvami

Vstup: Neorientovaný graf $G = (V, E)$, kde V je množina vrcholů a E je množina hran

Výstup: Odpověď na otázku, zda lze graf G obarvit třemi barvami. Tzn., že se ptáme, zda existuje zobrazení $c : V \to \{col1, col2, col3\}$ takové, že pro všechny $\{v1, v2\}$ patřících do E platí $c(v1) \neq c(v2)$.

Příklad: Na obrázku 4 můžeme vidět graf, který lze obarvit 3 barvami i s ukázkou jednoho možného takového obarvení.



Obrázek 4: Instance problému 3-CG [6]

2.2.4 Problém SUBSET-SUM

Název: Problém součtu podmnožiny

Vstup: Sekvence přirozených čísel a_1, a_2,..., a_n a přirozené číslo s

Výstup: Odpověď na otázku, zda existuje množina $I \subseteq 1, 2, ..., n$ taková, že $\sum_{i \in I} a_i = s$. Jinak řečeno, ptáme se, zda z dané (multi)množiny čísel je možné vybrat podmnožinu, jejíž součet je s.

Příklad s odpovědí *ANO*: sekvence tvořená čísly: 4, 7, 3, 4, 8 a číslo $s = 19$

Příklad s odpovědí *NE*: sekvence tvořená čísly: 4, 7, 3, 4, 8 a číslo $s = 20$

Poznámky:

- Pořadí čísel na vstupu a_1, a_2,..., a_n není důležité.
- Rozdíl mezi sekvencí a množinou je, že v množině se stejná čísla (prvky) neopakují, kdežto v sekvenci se může totéž číslo vyskytnout vícekrát.

2.2.5 Problém ILP

Název: Celočíselné lineární programování

Vstup: Celočíselná matice A a celočíselný vektor b

Výstup: Odpověď na otázku, zda existuje celočíselný vektor x, takový že $Ax < b$.

Příklad: $A = \begin{pmatrix} 4 & -2 & 5 \\ 1 & 0 & 1 \\ 3 & 1 & 0 \end{pmatrix}$ $b = \begin{pmatrix} 4 \\ -3 \\ 1 \end{pmatrix}$

Ptáme se tedy, zda existuje celočíselné řešení následující soustavy nerovnic:

$4x_1 - 2x_2 + 5x_3 \leq 4$
$x_1 + x_3 \leq -3$
$3x_1 + x_2 \leq 1$

Jedním z řešení soustavy je například: $x_1 = -4, x_2 = 1, x_3 = 1$, tj. $x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$

Pro tuto instanci je tedy odpověď ANO.

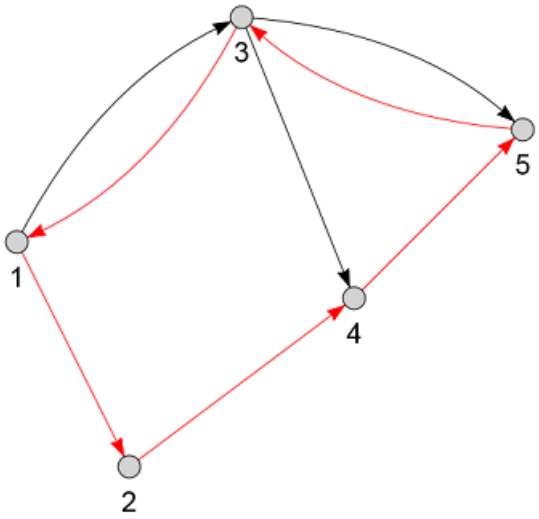
2.2.6 Problém HC

Název: Problém hamiltonovského cyklu

Vstup: Orientovaný graf G

Výstup: Odpověď na otázku, zda existuje v grafu G hamiltonovský cyklus. Tzn., že se ptáme, zda je v grafu G orientovaná kružnice procházející každý vrchol právě jednou.

Příklad: Na obrázku 5 můžeme v grafu vidět hamiltonovský cyklus skládající se z červených hran.



Obrázek 5: Hamiltonovský cyklus

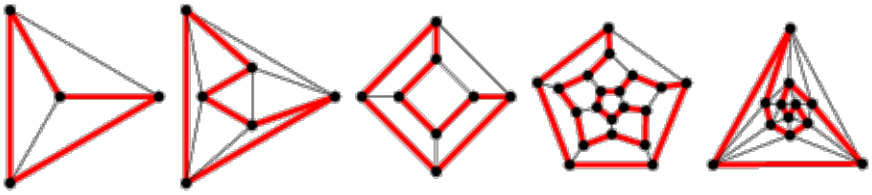
2.2.7 Problém HK

Název: Problém hamiltonovské kružnice

Vstup: Neorientovaný graf G

Výstup: Odpověď na otázku, zda existuje v grafu G hamiltonovská kružnice. Ptáme se tedy, jestli je v grafu G neorientovaná kružnice procházející každý vrchol právě jednou.

Příklad: Na obrázku 6 můžeme vidět v pěti různých grafech hamiltonovské kružnice skládající se z červených hran.



Obrázek 6: Příklady hamiltonovských kružnic [7]

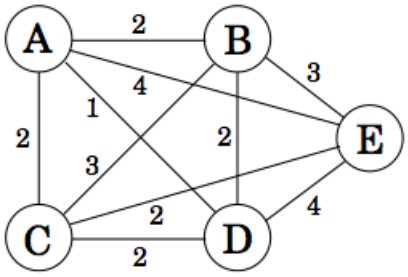
2.2.8 Problém TSP

Název: Problém obchodního cestujícího

Vstup: Úplný neorientovaný graf G s n vrcholy ("městy"), kde každé hraně (u, v) je přiřazeno celé kladné číslo $d(u, v)$ - vzdálenost mezi městy, a číslo L (limit).

Výstup: Odpověď na otázku, zda existuje v grafu G „okružní jízda“ dlouhá nejvýše L, což je uspořádání vrcholů v_1, v_2,..., v_n pro které platí: $d(v_1, v_2) + d(v_1, v_2) + ... + d(v_{n-1}, v_n) + d(v_n, v_1) \leq L$.

Příklad: Na obrázku 7 můžeme vidět úplný neorientovaný ohodnocený graf, pro který je odpověď ANO, pokud bude $L \geq 10$, například pro cestu vrcholy B, D, A, C, E, B. V opačném případě bude odpověď NE, protože nebude existovat „okružní jízda“ délky menší než 10.



Obrázek 7: Problém obchodního cestujícího [8]

2.3 Převody mezi NP-úplnými problémy

Tato podkapitola obsahuje konkrétní popis všech algoritmů převodů mezi NP-úplnými problémy, které byly implementovány v rámci této diplomové práce, včetně zdůvodnění jejich korektnosti.

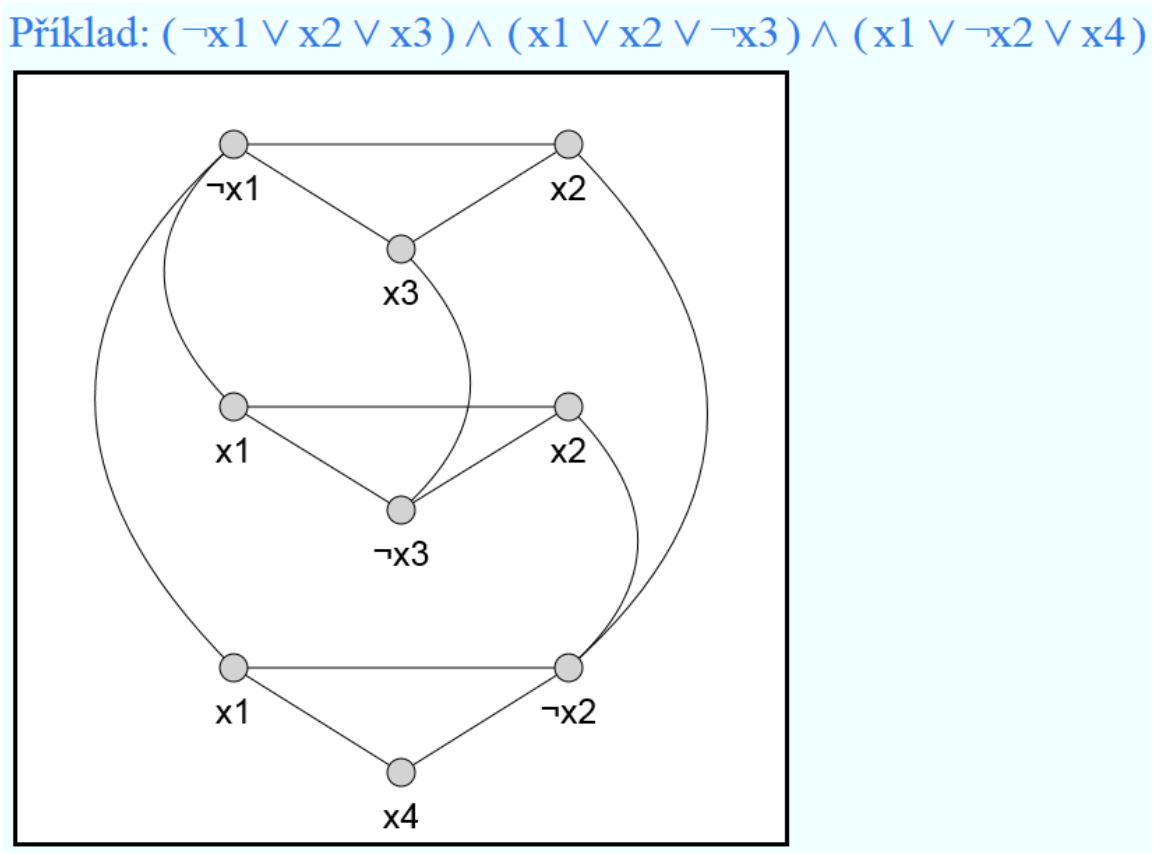
2.3.1 Převod 3-SAT na IS

Poznámka: Popis problému 3-SAT lze nalézt v kapitole 2.2.1 a popis problému IS v kapitole 2.2.2.

Převod Z definicí obou problémů vychází, že algoritmus bude muset převádět booleovskou formuli f v konjunktivní normální formě s k klauzulemi, kde v každé klauzuli jsou právě 3 literály na graf G a číslo n tak, aby graf G obsahoval nezávislou množinu složenou z n vrcholů právě tehdy, když je booleovská formule f splnitelná.

Popis algoritmu Prvním krokem algoritmu je vytvoření vrcholů grafu G a to tak, že literály ze vstupní formule f budou reprezentovat jednotlivé vrcholy grafu G. Poté se do grafu G přidají hrany mezi vrcholy reprezentující literály, které patří do stejné klauzule. Tím pádem nám v grafu G vzniknou trojúhelníky, protože v každé klauzuli jsou přesně 3 literály. Počet těchto

trojúhelníků bude potom počet klauzulí formule f. Poté se vloží hrany mezi vrcholy představující literály a jejich negace (pokud nejsou ve stejných klauzulích a tím pádem již spojeny). Tím by měl být graf G sestrojen. Jako požadovaný počet vrcholů pro nezávislou množinu (v obecném popisu převodu označený n) se zvolí počet klauzulí (v obecném popisu převodu označený k). Pro lepší pochopení, jak celá konstrukce vypadá, slouží obrázek 8.



Obrázek 8: Konstrukce grafu pro problém IS

Zdůvodnění korektnosti Aby byla vstupní formule f pravdivá v nějakém ohodnocení, tak musí každá klauzule při tomto ohodnocení obsahovat alespoň 1 pravdivý literál. Pokud v grafu G vybereme vrcholy reprezentující přesně tyto literály (vybereme pouze 1 pravdivý z každé klauzule), tak máme jistotu, že mezi sebou nejsou spojeny hranou (a tím pádem tvoří nezávislou množinu). Je to dáno tím, že každý vrchol se nachází v jiné klauzuli, čímž můžeme vyloučit hrany spojující literály ze stejné klauzule. Zbývají ještě hrany spojující literály a jejich negace, ale ty zase můžeme vyloučit díky tomu, že jsme vybrali pouze vrcholy reprezentující pravdivé literály, ale negace pravdivého literálu je nepravdivý literál, tudíž nemohl být vybrán. Žádné další hrany jsme algoritmem v grafu G nesestrojili a tím pádem víme, že nejsou vybrané vrcholy propojeny hranou - tvoří nezávislou množinu. Velikost této nezávislé množiny n je pak samo-

zřejmě rovna počtu klauzulí, jelikož jsme z každé klauzule vybrali právě 1 pravdivý literál (a tedy i 1 reprezentující vrchol).

Naopak, existuje-li nějaká nezávislá množina velikosti počtu klauzulí vstupní formule k, bude tato nezávislá množina nutně obsahovat jeden vrchol z každého trojúhelníku. Pokud ohodnocení proměnných zvolíme tak, aby literály odpovídající těmto vrcholům byly pravdivé, bude pravdivá celá formule. Protože navíc každý vrchol odpovídající nějaké proměnné je spojen s každým vrcholem odpovídajícím její negaci, nemůže nastat situace, že bychom při tvorbě ohodnocení museli nastavit na pravdivou současně proměnnou i její negaci.

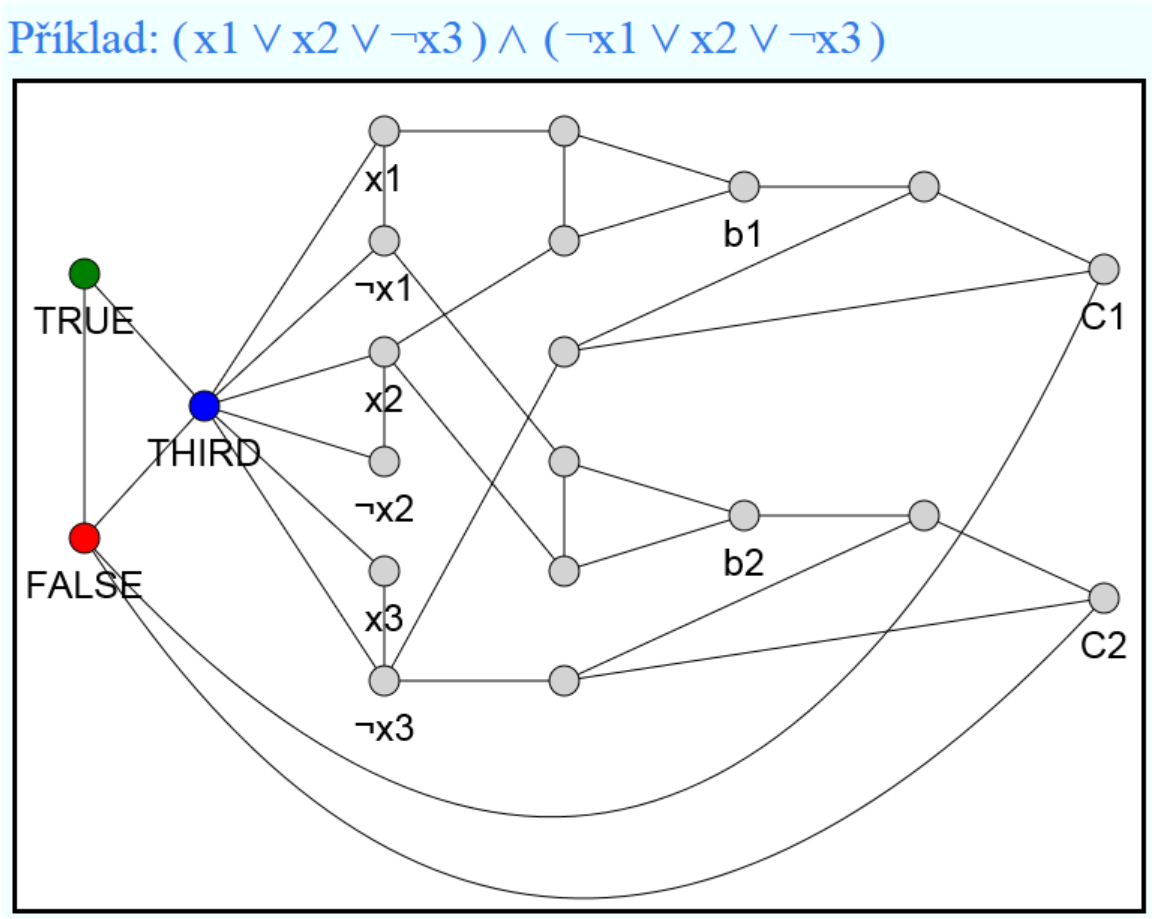
2.3.2 Převod 3-SAT na 3-CG

Poznámka: Popis problému 3-SAT lze nalézt v kapitole 2.2.1 a popis problému 3-CG v kapitole 2.2.3.

Převod Z definicí obou problémů vychází, že algoritmus bude muset převádět booleovskou formuli f v konjunktivní normální formě, kde v každé klauzuli jsou právě 3 literály, na graf G tak, aby byl graf G obarvitelný třemi barvami právě tehdy, když je booleovská formule f splnitelná.

Popis algoritmu Na začátku si vytvoříme 3 vrcholy různých barev a propojíme je hranami do trojúhelníku. Pro lepší pochopení je nazveme například $TRUE$ (zelený), $FALSE$ (červený) a $THIRD$ (modrý). Na tom, jaké zvolíme barvy, pochopitelně nezáleží, jde pouze o to, aby se lišily. Poté za každou proměnnou nacházející se ve formuli a také za její negaci vytvoříme 2 vrcholy a ty propojíme mezi sebou a také s vrcholem $THIRD$. Tím zajistíme, že proměnná bude muset mít opačnou barvu než její negace a že pro každý z těchto dvou vrcholů bude na výběr jen ze dvou barev - zelené a červené.

Nakonec vytvoříme pro všechny klauzule následující konstrukci. Vytvoříme si 3 pomocné vrcholy, které mezi sebou propojíme. První z nich ještě propojíme s prvním literálem z klauzule, druhý s druhým literálem z klauzule a třetí si nazveme například b. Obdobnou konstrukci, co jsme sestrojili s prvními dvěma literály z klauzule, uděláme s vrcholem b a třetím literálem z klauzule. Jen s tím rozdílem, že poslední pomocný vrchol nenazveme b, ale například C (podle slova clause). Ten nakonec ještě propojíme s vrcholem $FALSE$. Jak už bylo řečeno, tak toto provedeme pro všechny klauzule formule f. Konstrukce grafu G je poté hotová. Pro lepší pochopení, jak celá konstrukce vypadá, slouží obrázek 9.



Obrázek 9: Konstrukce instance problému 3-CG

Zdůvodnění korektnosti Aby byla vstupní formule pravdivá v nějakém ohodnocení, tak musí každá klauzule při tomto ohodnocení obsahovat alespoň 1 pravdivý literál. Pokud bude literál pravdivý, obarvíme jej zelenou barvou, v opačném případě barvou červenou. Pokud bude alespoň 1 z prvních 2 literálů v klauzuli pravdivý (tedy zelený), tak budeme moci pomocný vrchol s ním propojený obarvit červeně a druhý pomocný vrchol modře. Tím pádem budeme moci vrchol b obarvit zeleně. Na základě stejného principu obarvíme i další konstrukci. Tedy principu, že bude alespoň 1 vstupní vrchol zelený – což bude buď vrchol b, pokud bude pravdivý alespoň 1 z prvních 2 literálů, anebo vrchol reprezentující třetí literál v klauzuli, protože alespoň 1 musí být pravdivý. Tím pádem vrchol C bude také zelený. Jak bylo řečeno na začátku, tak aby byla vstupní formule pravdivá, tak každá klauzule musí mít pravdivý alespoň 1 literál, tím pádem bude také vrchol C zelený pro každou klauzuli a nedostaneme se tím do konfliktu s vrcholem $FALSE$.

Pokud je formule f nesplnitelná, tak každé možné ohodnocení způsobí nepravdivost všech literálů alespoň jedné klauzule. To znamená, že každé možné obarvení vrcholů zastupující proměnné a jejich negace obarví vrcholy odpovídající literálům alespoň jedné klauzule všechny červeně. Pro

pomocné vrcholy této klauzule potom nutně musí být použita barva zelená a modrá a na vrchol b tak zůstane barva červená. Podle stejného principu bude muset být červený i vrchol C. Ten je ale spojený s vrcholem $FALSE$ a tudíž graf G nelze správně obarvit třemi barvami, pokud není formule f splnitelná. Tím je zdůvodnění korektnosti hotovo.

2.3.3 Převod 3-SAT na SUBSET-SUM

Poznámka: Popis problému 3-SAT lze nalézt v kapitole 2.2.1 a popis problému SUBSET-SUM v kapitole 2.2.4.

Převod Z definicí obou problémů vychází, že redukční algoritmus bude převádět booleovskou formuli f v konjunktivní normální formě, kde v každé klauzuli jsou právě 3 literály na multimnožinu přirozených čísel X a přirozené číslo t tak, aby v multimnožině X existovala podmnožina čísel, jejichž součet by se rovnal přirozenému číslu t právě tehdy, když je booleovská formule f splnitelná.

Popis algoritmu Čísla v multimnožině X budeme vytvářet po jednotlivých číslicích daného čísla zapsaného (pro jednodušší pochopení) v číselné soustavě o základu $d = 10$. Instanci problému SUBSET-SUM budeme vytvářet jako dvojrozměrnou tabulku, kde jednotlivé řádky budou čísla v multimnožině X a poslední řádek bude věnován našemu součtu t.

Tabulku zkonstruujeme následovně:

- Pro každou proměnnou x_i ve vstupní formuli f budou v tabulce (a tedy i v multimnožině X) řádky y_i a z_i, kde i bude index proměnné.
- Pro každou klauzuli nacházející se ve formuli budeme potřebovat pomocné řádky g_j a h_j, kde j bude index klauzule.
- Pro každou proměnnou nacházející se ve formuli budeme potřebovat sloupec, který si označíme stejně jako tuto proměnnou.
- Pro každou klauzuli nacházející se ve formuli budeme potřebovat sloupec, který si označíme c_j, kde j bude index klauzule.
- Poslední řádek bude věnován sumě t.

A hodnoty do tabulky vyplníme následovně:

- $(y_i, x_j), (z_i, x_j) - 1$ pokud $i = j$, jinak 0
- $(y_i, c_j) - 1$ pokud c_j obsahuje proměnnou x_i, jinak 0

- $(z_i, c_j) - 1$ pokud c_j obsahuje proměnnou $\neg x_i$, jinak 0
- $(g_i, x_j), (h_i, x_j) - 0$
- $(g_i, c_j), (h_i, c_j) - 1$ pokud $i = j$, jinak 0
- $(t, x_j) - 1$
- $(t, c_j) - 3$

Pro lepší pochopení, jak celá konstrukce vypadá, slouží obrázek 10. Výsledná multimnožina X vyplývající z tabulky je {100110, 1000001, 0100011, 0100100, ..., 0000001}, a číslo t je 1111333.

Příklad: $(x1 \vee \neg x2 \vee x3) \wedge (x1 \vee x2 \vee x3) \wedge (\neg x1 \vee x2 \vee x4)$

	x1	x2	x3	x4	c1	c2	c3
y1	1	0	0	0	1	1	0
z1	1	0	0	0	0	0	1
y2	0	1	0	0	0	1	1
z2	0	1	0	0	1	0	0
y3	0	0	1	0	1	1	0
z3	0	0	1	0	0	0	0
y4	0	0	0	1	0	0	1
z4	0	0	0	1	0	0	0
g1	0	0	0	0	1	0	0
h1	0	0	0	0	1	0	0
g2	0	0	0	0	0	1	0
h2	0	0	0	0	0	1	0
g3	0	0	0	0	0	0	1
h3	0	0	0	0	0	0	1
t	1	1	1	1	3	3	3

Obrázek 10: Sestrojená tabulka pro problém SUBSET-SUM

Zdůvodnění korektnosti Je-li formule f splnitelná, existuje nějaké ohodnocení, při kterém je pravdivá. Čísla z množiny X vybereme podle tohoto ohodnocení následovně:

- Ponecháme řádek y_i, pokud je odpovídající proměnná $true$.
- Ponecháme řádek z_i, pokud je odpovídající proměnná $false$.
- Ponecháme řádek g_j, pokud počet pravdivých literálů v c_j je nejvýše 2.

- Ponecháme řádek h_j, pokud počet pravdivých literálů v c_j je 1.

Ať už je ohodnocení vstupní formule f jakékoliv, tak za každou proměnnou x_i vybereme buď řádek y_i anebo řádek z_i (to vychází z prvních dvou pravidel pro výběr čísel z multimnožiny, která jsou popsána výše). V ostatních řádcích je vždy hodnota 0, tudíž součet bude pro všechny sloupce určené proměnným x_i vždy 1. Vybrané řádky x_i a y_i musí mít ve sloupcích c_j určeným jednotlivých klauzulím vstupní formule f, vždy součet v rozmezí z 0-3. Tato hodnota je dána počtem pravdivých literálů (při daném ohodnocení) ve vstupní formuli f.

Jak už bylo zmíněno u jiných zdůvodnění korektnosti, tak aby byla vstupní 3-SAT formule f pravdivá (má pravdivé ohodnocení), tak musí každá klauzule obsahovat alespoň 1 pravdivý literál. Pokud tedy bude vstupní formule f pravdivá (tedy každý sloupec c_j, bude mít průběžný součet alespoň 1), tak vždy můžeme doložit díky řádkům g_j a h_j na konečný součet 3 v daném sloupci c_j (máme v těchto řádcích dvě jedničky na dorovnání). A tím pádem pokud bude vstupní formule f pravdivá, tak vždy najdeme podmnožinu čísel v naší zkonstruované multimnožině X takovou, aby její součet byl roven t.

Jestliže v množině X bude nalezena podmnožina dávající součet t, určitě v ní je přesně jedna hodnota z každé dvojice y_i, z_i, protože každá proměnná je buď $true$ anebo $false$. Můžeme tedy vytvořit ohodnocení formule f, které dá hodnotu $true$ proměnným, pro které bylo vybráno y a $false$, kde bylo vybráno z. Protože vybrané hodnoty z množiny X dávají na pozicích odpovídajícím sloupcům c_j součet 3 a jen součet 2 je dosažitelný pomocí čísel z řádků g_j a h_j, určitě do tohoto součtu na každé pozici c_j přispívá nějaká vybraná proměnná y_i nebo z_i. To ale znamená, že příslušný pravdivý literál se nachází v j-té klauzuli. Protože to platí pro všechny pozice c_j, je v každé klauzuli alespoň jeden pravdivý literál a formule je tedy v tomto vytvořeném ohodnocení pravdivá.

2.3.4 Převod 3-SAT na ILP

Poznámka: Popis problému 3-SAT lze nalézt v kapitole 2.2.1 a popis problému ILP v kapitole 2.2.5.

Převod Z definicí obou problémů vychází, že algoritmus bude muset převádět booleovskou formuli f v konjunktivní normální formě, kde v každé klauzuli jsou právě 3 literály, na soustavu lineárních nerovnic takovou, že tato bude mít řešení v oboru celých čísel právě tehdy, když je booleovská formule f splnitelná.

Popis algoritmu

- Nejdříve si určíme zástupné neznámé za proměnné vstupní formule f. Řekněme, že každé proměnné x_i bude odpovídat neznámá x'_i.
- Poté začneme konstruovat sestavu nerovnic. Pro každou neznámou x'_i přidáme do soustavy dvojici nerovnic $x'_i \geq 0$ a $x'_i \leq 1$.
- Za každou klauzuli tvaru $(L_1 \wedge L_2 \wedge L_3)$, kde L_i jsou jednotlivé literály, přidáme do soustavy nerovnic nerovnici: $f_1 + f_2 + f_3 \geq 1$, kde:
  - $f_i = x'_i$, pokud $L_i = x'_i$
  - $f_i = (1 - x'_i)$, pokud $L_i = \neg x'_i$
- Tím je soustava nerovnic hotová, jen ji je ještě potřeba upravit (jednoduchými aritmetickými operacemi) do tvaru: $c_1 * x'_1 + c_2 * x'_2 + ... + c_n * x'_n \leq d$, kde c_1, c_2,..., c_n a d jsou konstanty.
- Po těchto úpravách máme již soustavu nerovnic v požadovaném tvaru a můžeme ji zapsat maticovým způsobem (viz obrázek 11), abychom mohli říci, že vstup problému ILP je takový, jaký si tento problém vyžaduje.

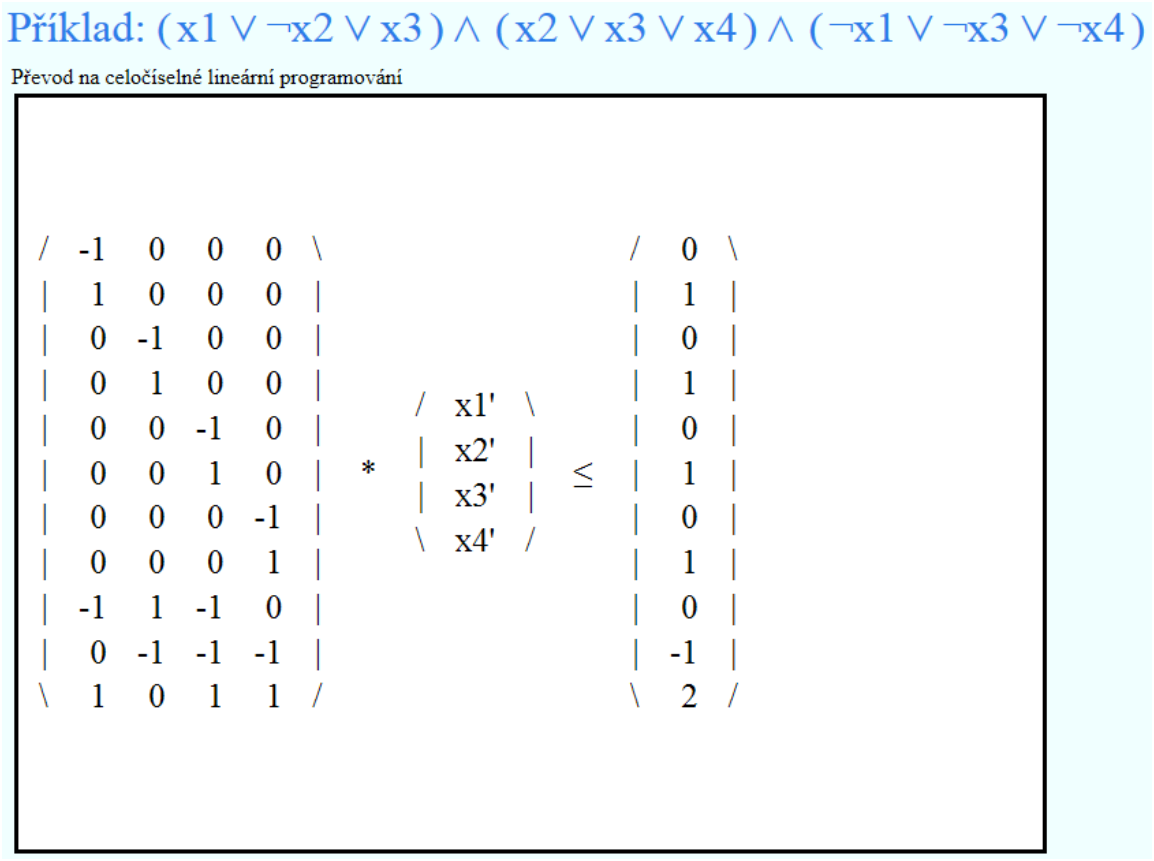
Zdůvodnění korektnosti Jelikož soustava nerovnic obsahuje pro každou neznámou x'_i nerovnice $x'_i \geq 0$ a $x'_i \leq 1$, tak musí být řešení celé soustavy (pokud existuje) takové, že jednotlivé neznámé x'_i můžou mít hodnoty pouze 0 anebo 1. Tudíž dle ohodnocení proměnných x_i vstupní formule f, budou neznámým v soustavě nerovnic odpovídat tyto hodnoty:

- $x'_i = 0$, když $[x_i]_v = false$
- $x'_i = 1$, když $[x_i]_v = true$

Tímto nám nerovnice prvního typu určily rámec, ve kterém můžeme operovat při hledání řešení u druhých rovnic. Díky tomu že tyto nerovnice jsou tvaru $f_1 + f_2 + f_3 \geq 1$, kde $f_i = x'_j$, pokud $L_i = x_j$ nebo $f_i = (1 - x'_j)$, pokud $L_i = \neg x_j$, víme, že nezáleží na tom, zda má literál L_i tvar x_j nebo $\neg x_j$, protože pokaždé platí, že:

- $f'_i = 1$, když $[x_i]_v = true$
- $f'_i = 0$, když $[x_i]_v = false$

Jinými slovy hodnota levé strany nerovnice $(f_1 + f_2 + f_3)$ má hodnotu počtu literálů v klauzuli $(L_1 \wedge L_2 \wedge L_3)$, které mají ohodnocení $true$. A jak víme z předchozích převodů, tak aby byla vstupní 3-SAT formule f pravdivá (má pravdivé ohodnocení), tak musí každá klauzule obsahovat



Obrázek 11: Maticový zápis převedené soustavy nerovnic

alespoň 1 pravdivý literál. Tím vlastně splníme pravou stranu nerovnice (≥ 1) právě tehdy, když bude vstupní formule f splnitelná.

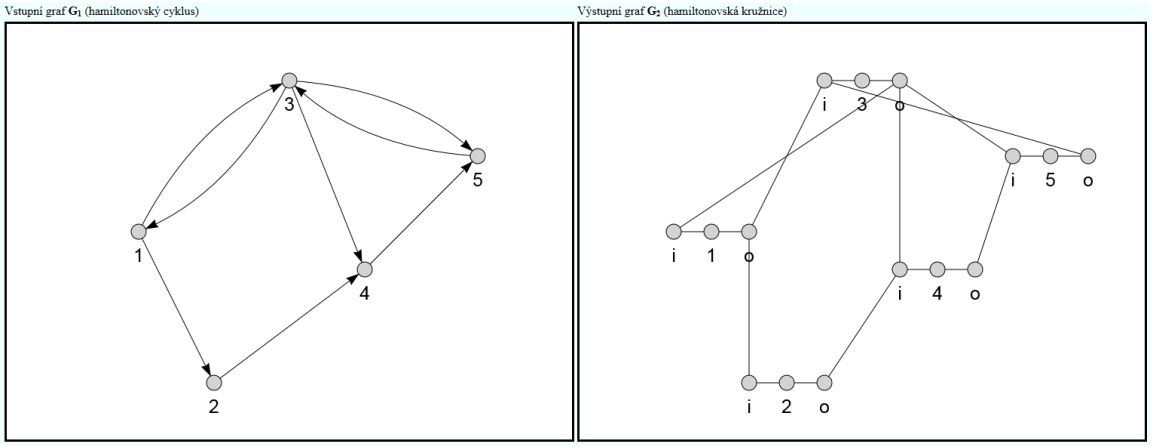
2.3.5 Převod HC na HK

Poznámka: Popis problému HC lze nalézt v kapitole 2.2.6 a popis problému HK v kapitole 2.2.7.

Převod Z definicí obou problémů vychází, že algoritmus bude převádět orientovaný graf G_1 na neorientovaný graf G_2 a to tak, že ve vstupním orientovaném grafu G_1 bude existovat hamiltonovský cyklus právě tehdy, když ve vytvořeném výstupním neorientovaném grafu G_2 bude existovat hamiltonovská kružnice.

Popis algoritmu Výstupní neorientovaný graf G_2 vytvoříme tak, že nejprve za každý vrchol vstupního orientovaného grafu G_1 vytvoříme trojici vrcholů spojených dvěma neorientovanými hranami. Prostřední vrchol z trojice bude suplovat příslušný vrchol vstupního grafu G_1 a krajní

vrcholy budou pomocné pro daný vrchol - jako vstupní a výstupní vrchol. Poté za každou orientovanou hranu vstupního grafu G_1 z vrcholu A do vrcholu B dáme do výstupního grafu G_2 neorientovanou hranu spojující trojice odpovídající vrcholům A a B. Tato hrana bude konkrétně spojovat výstupní vrchol spojený s vrcholem A a vstupní vrchol spojený s vrcholem B. Pro lepší pochopení, jak celá konstrukce vypadá, slouží obrázek 12, kde jsou výstupní vrcholy ve výstupním G_2 grafu označeny jako o (output) a vstupní vrcholy jako i (input).



Obrázek 12: Převod vstupu HC na vstup HK

Zdůvodnění korektnosti Algoritmus nemůže pouze jednoduše převést orientovaný graf G_1 na neorientovaný graf G_2 způsobem, že zkrátka zruší orientaci jednotlivých hran (a popřípadě ještě smaže potenciálně vzniknuvší násobné hrany), jelikož by bylo možné, že by orientovaný graf G_1 neobsahoval hamiltonovský cyklus, ale neorientovaný graf G_2 by obsahoval hamiltonovskou kružnici (nebyla by zachována odpověď NE na oba problémy). Mohlo by totiž nastat, že smazáním orientace hran vznikne v grafu G_2 hamiltonovská kružnice, i když v původním grafu G_1 nebyl hamiltonovský cyklus.

Pokud bychom zvolili možnost vytvořit za každý vrchol grafu G_1 2 vrcholy (vstupní a výstupní), tak by opět mohl vzniknout případ, kdy by ve vstupním grafu G_1 nebyl hamiltonovský cyklus, ale v grafu G_2 vznikla hamiltonovská kružnice. Neměli bychom totiž jistotu, zda je v ní příslušný vrchol pouze jednou anebo dvakrát. Jinak řečeno hamiltonovská kružnice by ve výstupním grafu G_2 mohla vzniknout právě díky tomu, že jsme (na rozdíl od vstupního grafu G_1) přidali možnost využít pro hamiltonovskou kružnici 1 vrchol jakoby dvakrát (jednou jako vstupní a jednou jako výstupní vrchol).

Tím že jsme ale za každý vrchol vytvořili 3 vrcholy, jsme tyto potenciální nesrovnalosti v převodu odstranili. Díky vstupním a výstupním vrcholům jsme nemuseli mazat žádné původní orientované hrany - nevznikly žádné násobné. Prostřední vrchol nám zase dává stoprocentní jistotu, že každý původní vrchol bude pro hamiltonovskou kružnici využit pouze jednou, protože hamilto-

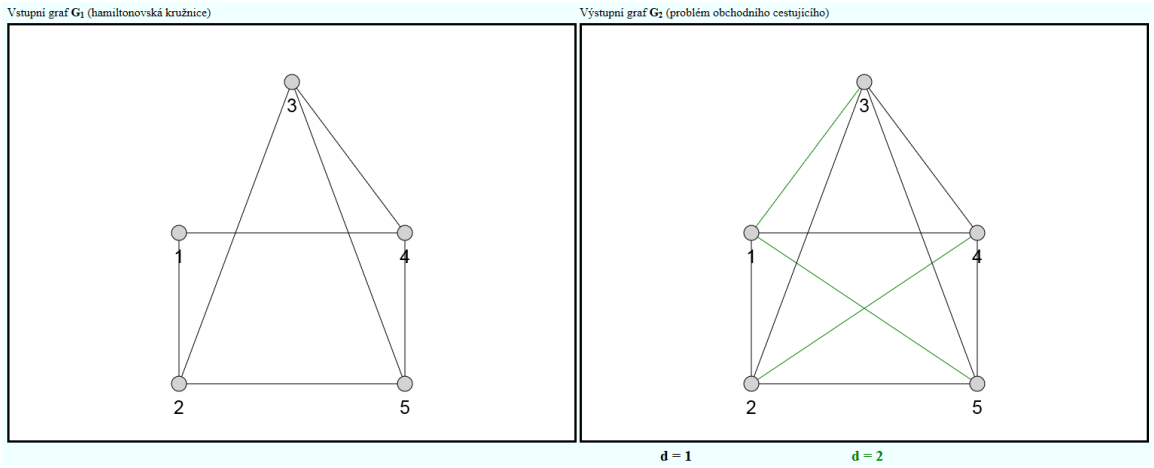
novská kružnice jím může projít pouze přes vstupní a výstupní vrchol příslušného původního vrcholu.

2.3.6 Převod HK na TSP

Poznámka: Popis problému HK lze nalézt v kapitole 2.2.7 a popis problému TSP v kapitole 2.2.8.

Převod Z definicí obou problémů vychází, že algoritmus bude převádět neorientovaný graf G_1 na úplný neorientovaný graf G_2 a číslo L. To tak, že ve vstupním neorientovaném grafu G_1 bude existovat hamiltonovská kružnice právě tehdy, když ve výstupním úplném neorientovaném grafu G_2 bude existovat „okružní jízda“ délky nejvýše L.

Popis algoritmu Algoritmus pro vytvoření instance problému obchodního cestujícího je v podstatě velice jednoduchý. Zkrátka vytvoříme úplný graf G_2 a to tak, že okopírujeme všechny vrcholy ze vstupního grafu G_1 a vložíme je do grafu G_2. To stejné provedeme s hranami, akorát že každou takto okopírovanou hranu ze vstupního grafu G_1 ohodnotíme ve výstupním grafu G_2 délkou $d(u, v) = 1$. Jelikož výstupní graf G_2 má být úplný, tak v něm ještě vytvoříme hrany mezi všemi vrcholy, mezi kterými ještě žádná není a ohodnotíme ji délkou $d(u, v) = 2$. Tím je převod vstupů obou problémů hotový. Na obrázku 13 je pro lepší pochopení příklad takovéto konstrukce.



Obrázek 13: Převod vstupu HK na vstup TSP

Zdůvodnění korektnosti Pokud bude ve vstupním grafu G_1 hamiltonovská kružnice délky L (počet vrcholů v obou grafech), tak můžeme pro „okružní jízdu“ ve výstupním grafu G_2 použít pouze hrany délky $d(u, v) = 1$ (a tudíž délka „okružní jízdy“ bude právě L). Jak víme z

popisu konstrukce, tak tyto hrany jsou v obou grafech. Tím pádem víme, že pokaždé když bude ve vstupním grafu G_1 hamiltonovská kružnice, tak ve výstupním grafu G_2 najdeme „okružní jízdu“ délky L a převod je tedy korektní pro odpověď ANO.

Pokud by ve vstupním grafu G_1 nebyla hamiltonovská kružnice, tak ve výstupním grafu G_2 nemůžeme nalézt „okružní jízdu“ délky L, protože bychom pro její nalezení museli použít nějakou hranu, která se nenachází ve vstupním grafu G_1 a tudíž je její délka $d(u, v) = 2$ - čímž bychom překročili limit L. Takže pokud ve vstupním grafu G_1 nebude hamiltonovská kružnice, tak ve výstupním grafu nikdy nenalezneme „okružní jízdu“ délky L a náš převod je tudíž korektní i pro odpověď NE.

3 Analýza požadavků

Tato kapitola se bude věnovat analýze požadavků na nový server pro podporu výuky teoretické informatiky. V této analýze jsem nevycházel pouze ze samotného zadání diplomové práce, ale také z průběžných požadavků vedoucího práce a svých vlastních (avšak skromných) zkušeností a předpokladů.

3.1 Funkční požadavky

PROČ nový systém Tento systém vznikl pro podporu výuky v oblasti teoretické informatiky. Jeho konkrétní zaměření je na oblast NP-úplných problémů, zejména tedy na převody mezi nimi. Hlavní motivací byla možnost spustit výpočet jednotlivých převodů a na zobrazeném průběhu výpočtu lépe pochopit principy fungování těchto algoritmů.

K ČEMU bude systém sloužit Primární funkcí systému není nejefektivnější implementace algoritmů (ve smyslu jak bývá zvykem), ale taková, která umožní postupně zobrazovat jednotlivé kroky tak, aby uživatel mohl co nejlépe pochopit, jak vlastně fungují a co se vstupem vykonávají. Bude tedy sloužit jako nástroj pro efektivnější pochopení převodů mezi NP-úplnými problémy.

KDO bude se systémem pracovat Systém je určen především pro studenty předmětů z oblasti teoretické informatiky, jejichž náplní jsou NP-úplné problémy a převody mezi nimi, popřípadě pro samotné pedagogy jako pomůcka v rámci výuky této oblasti. V neposlední řadě může být systém využíván širokou veřejností, pokud bude aplikace nasazená na veřejně přístupný server. V podstatě se totiž jedná o internetové stránky, na které není nutná žádná registrace či přihlašování.

VSTUPY a VÝSTUPY Vstupy systému budou instance jednotlivých vstupních NP-úplných problémů naimplementovaných převodů. Pro každý převod bude moci uživatel sám zadat vstupní data a budou zde také předdefinované ukázkové vstupy, aby uživatel nemusel nutně sám zadávat tato vstupní data. Jak už bylo řečeno výše, cílovým výstupem není ani tak samotný výstup celého algoritmu jako takový (samozřejmě je ale zobrazen také), nýbrž zejména zobrazení průběhu těchto převodů po jednotlivých krocích s odpovídajícím popisem.

3.2 Nefunkční požadavky

Výsledným softwarovým dílem je internetová aplikace, která původně vznikla jako ASP.NET projekt, avšak postupem času byla předělána na klasické čisté HTML stránky. To znamená, že k jejímu nasazení již není nutný server s podporou technologie ASP.NET, tak jako tomu bylo původně zamýšleno. Tato aplikace by měla korektně fungovat minimálně v nejpoužívanějších prohlížečích, jako jsou MS Internet Explorer, Google Chrome a Mozilla Firefox.

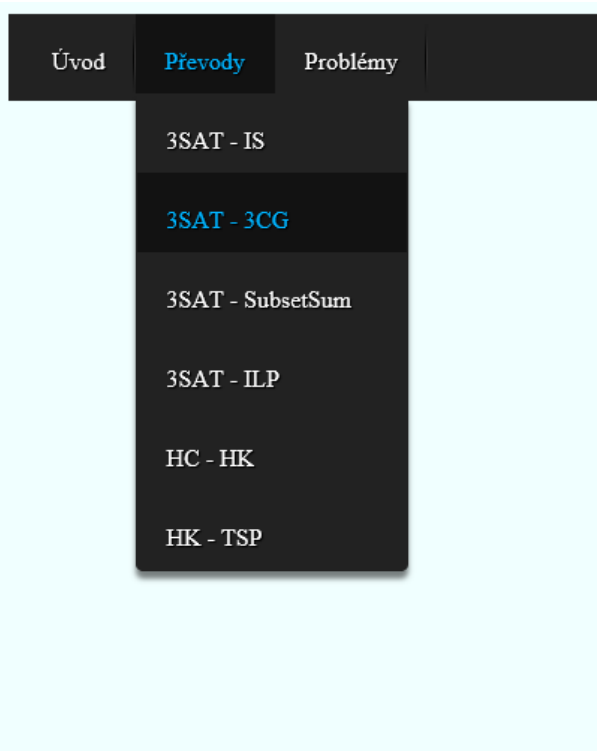
Pozornost bude kladena zejména na korektnost, srozumitelnost a přehlednost výpočtů jednotlivých algoritmů, ale také na plynulost a rychlou odezvu stránek.

4 Uživatelské rozhraní

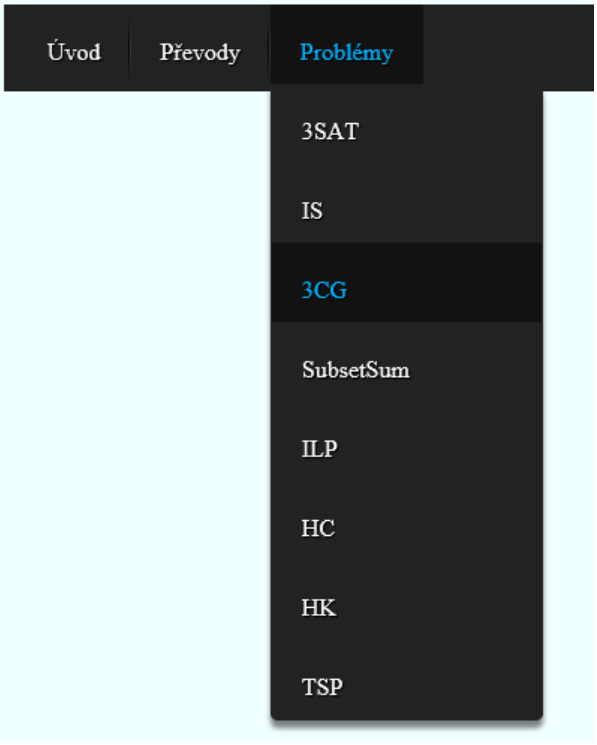
Tato kapitola se věnuje základnímu popisu grafického uživatelského rozhraní stránek. Bude zde popsáno, co se kde nachází, základní rozložení komponent apod.

4.1 Menu

Základní menu se na každé stránce webové aplikace nachází v horní liště. Skládá se ze tří základních záložek. První z nich je Úvod, kde jsou pouze základní informace. Další záložka Převody je už zajímavější. Je zde rolovací menu, kde se nachází dalších 6 položek. Pod každou z těchto položek se pak skrývá jedna animace převodu. Pod poslední záložkou Problémy se nachází 8 položek, kde každá představuje jednotlivý NP-úplný problém, s kterým aplikace pracuje. Jedná se vlastně o odkazy na prezentace skládající se z několika málo HTML slajdů, kde je vždy blíže popsán konkrétní problém.



Obrázek 14: Menu pro převody

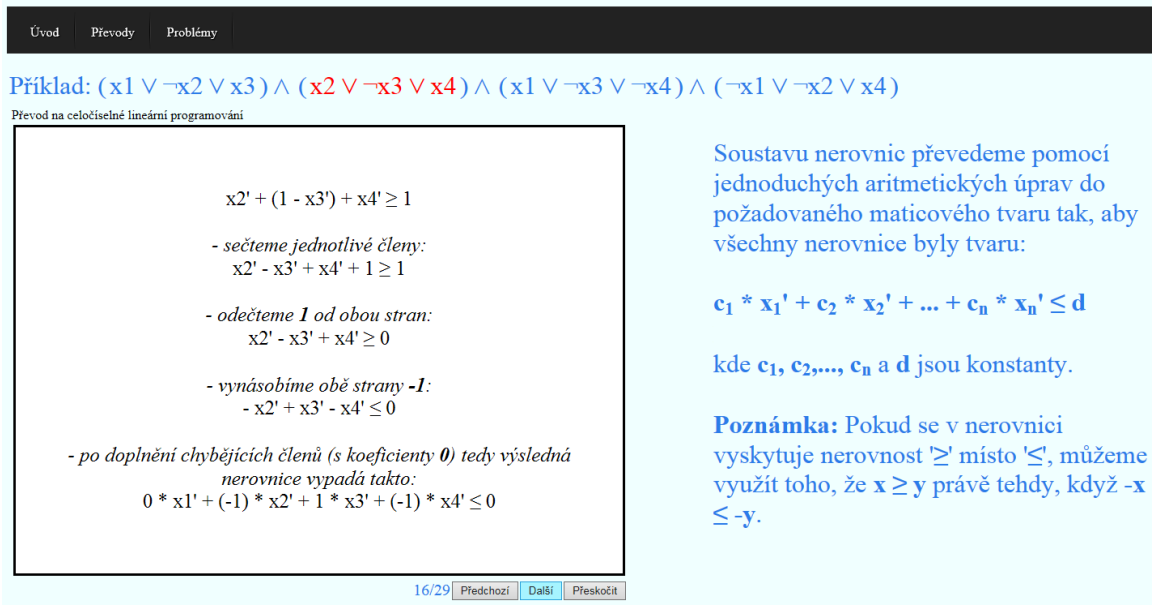


Obrázek 15: Menu pro problémy

4.2 Převody

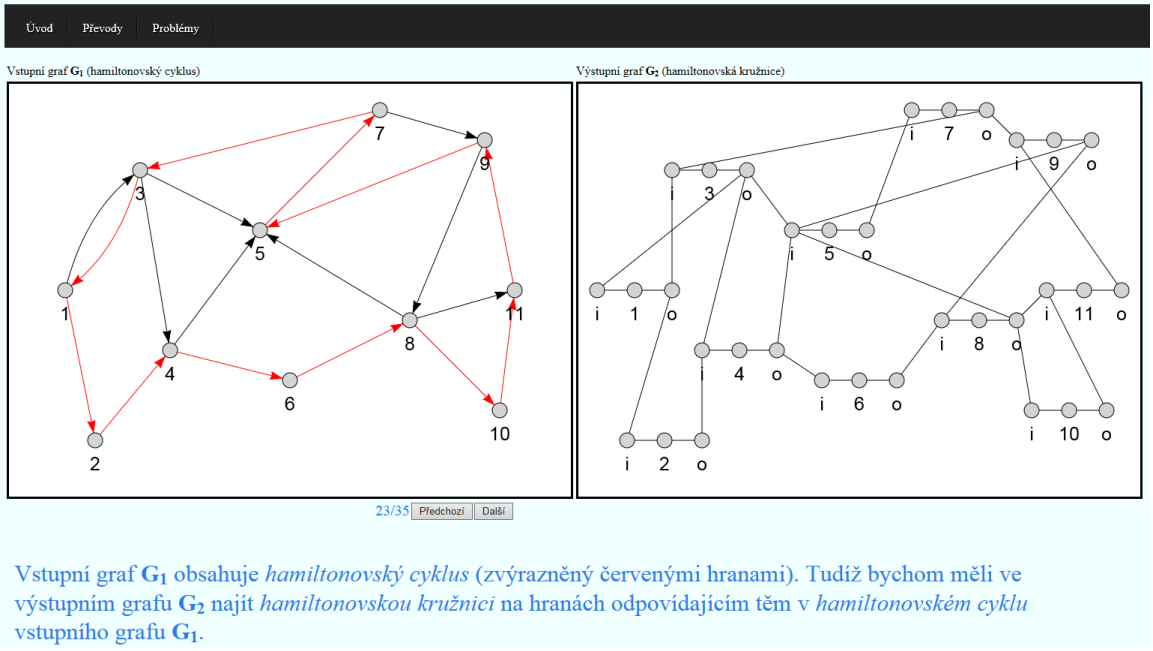
Jelikož je každý převod jiný a má odlišné vstupy a výstupy, tak ne pro každý převod může být rozložení potřebných komponent stejné. Například převod grafu na graf vyžaduje trochu jiné komponenty a tedy i jejich rozvržení a prostor než převod 3-SAT formule na graf. U převodů se mi v podstatě podařilo zachovat 2 skupiny zhruba stejných GUI.

První skupinou jsou převody, kde je vstupem formule, která si na stránce nevyžaduje tolik prostoru. Tento vstup (ať už editovatelný, či již uzamčený) je hned na začátku, tedy v levé horní části. Jsou zde všechny komponenty potřebné pro vstup, ať už textová pole, nabídky, tlačítka atd. Pod tímto vstupem je pak dole vlevo vyobrazena postupná transformace na výstupní problém, ať už ve formě grafu, tabulky, či textu. Pod touto komponentou jsou ještě tlačítka pro posouvání se v animaci převodu, ať už dozadu, dopředu, přeskočit apod. a je zde také zobrazen aktuální krok a celkový počet kroků dané simulace. Napravo od této části se pak nachází popis aktuálního kroku algoritmu.



Obrázek 16: První typ GUI pro převody

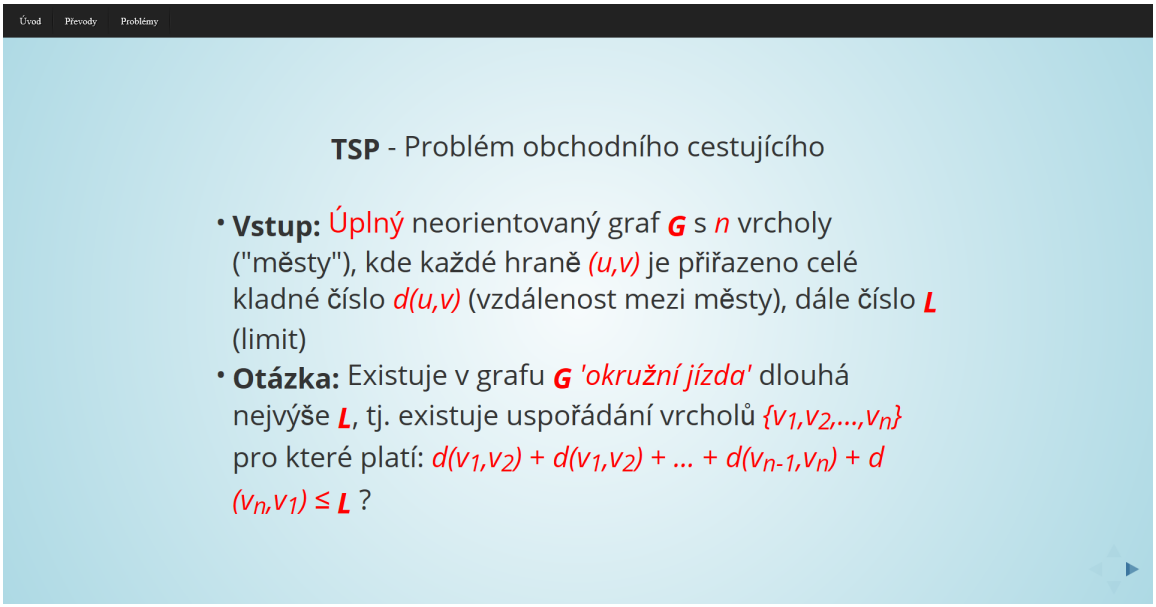
Do druhé skupiny patří převody, jejichž vstupními i výstupními problémy jsou grafy. Tudíž bylo potřeba vložit na stránku dvě stejně velké komponenty a tomu přizpůsobit vzhled stránky. Zvolil jsem tedy, že vstupní problém bude vlevo a výstupní vpravo, jak se logicky nabízelo, s tím že ovládací tlačítka pro krokování a informace o krocích nechám zhruba ve stejné pozici jako u první skupiny, což je vpravo pod kontejnerem pro vstupní problém (který je vlevo). Pro samotný popis animace převodu pak zbylo jediné místo, a to dole pod oběma problémy tak, aby se popis ještě vešel do okna klasických obrazovek, bez nutnosti rolovat stránku dolů.



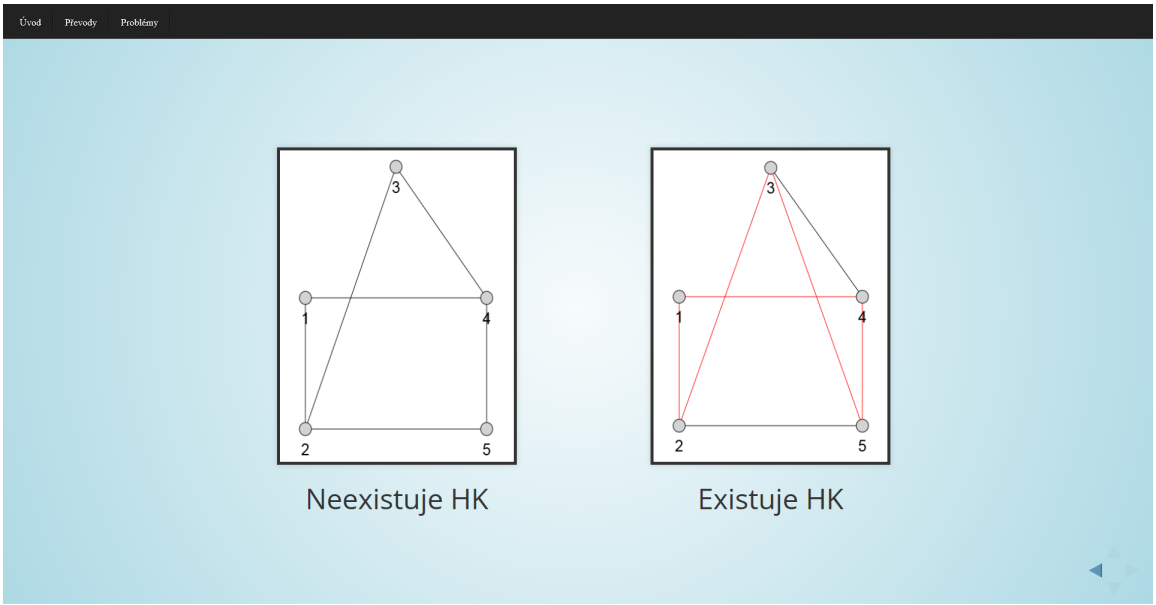
Obrázek 17: Druhý typ GUI pro převody

4.3 Problémy

Stránky nacházející se pod položkami v záložce Problémy jsou věnované teoretickým popisům jednotlivých problémů. Jejich obsah je ze značné části podobný obsahu z kapitoly 2.2. Tyto stránky jsou vytvořeny jako webové prezentace. Tyto prezentace jsem vytvořil pomocí frameworku pro HTML prezentace reveal.js [9]. Každá stránka obsahuje několik málo slajdů, které se mohou posouvat buď šipkami na klávesnici anebo navigačními šipkami v pravém dolním rohu. První stránka vždy obsahuje základní definici problému, poté následují slajdy s ukázkami příkladů, pro které je odpověď na otázku daného problému *ANO* a pro které *NE*. Text těchto slajdů je pak obohacen o styly (barvy, tučně, kurzíva, dolní indexy atd.) podle potřeby, kdy to přišlo vhod tak, aby uživatel mohl co nejjednodušeji a nejrychleji pochopit daný problém.



Obrázek 18: Slajd s definicí problému



Obrázek 19: Slajd s instancemi problému

5 Implementace

Tato kapitola popisuje nejen technologie, postupy a popis implementace výsledné aplikace, ale z části také postupný vývoj a změny v řešení, pokud byly zapotřebí.

5.1 Team Foundation Server

Tato aplikace byla vyvíjena ve vývojovém prostředí Microsoft Visual Studio 2015. Zhruba po první pětině z celkového procesu jsem se rozhodl pro organizování a ukládání mého postupu na této diplomové práci využít pomocného nástroje TFS (Team Foundation Server). A to ze dvou důvodů. Prvním bylo, že jsem cítil potřebu si rozvrhnout práci na jednotlivé kategorie a úkoly, jelikož jsem na to zvyklý ze zaměstnání a vím, že to pomáhá k efektivitě samotné práce. Dalším v podstatě mnohem důležitějším důvodem byla nutnost ukládat zdrojové soubory na co nejbezpečnější záložní místo a hlavně udržovat si jednotlivé verze programu. K oběma těmto požadavkům je TFS ideální.

Celé to fungovalo tak, že jsem si po každé konzultaci s vedoucím práce přetavil poznámky z konzultace do nejrůznějších kategorií a jednotlivých úkolů (tzv. tasků) v TFS, kde mohou být přehledně zobrazeny. Úkoly jsem přidával i v průběhu vývoje mimo konzultace, jak mě napadaly další nedostatky a vylepšení tak, aby vše bylo organizované a na nic bych nezapomněl. Každý úkol anebo několik málo úkolů (programátorských) jsem vždy přes Visual Studio, které je s TFS propojené, nahrál na svůj TFS a měl tak možnost v jednotlivých verzích na TFS sledovat změny ve zdrojových souborech a lépe tak kontrolovat svou práci a jednotlivé úkoly. Tento postup dokáže zabránit mnoha chybám a ty vzniknuvší pomáhá rychleji najít.

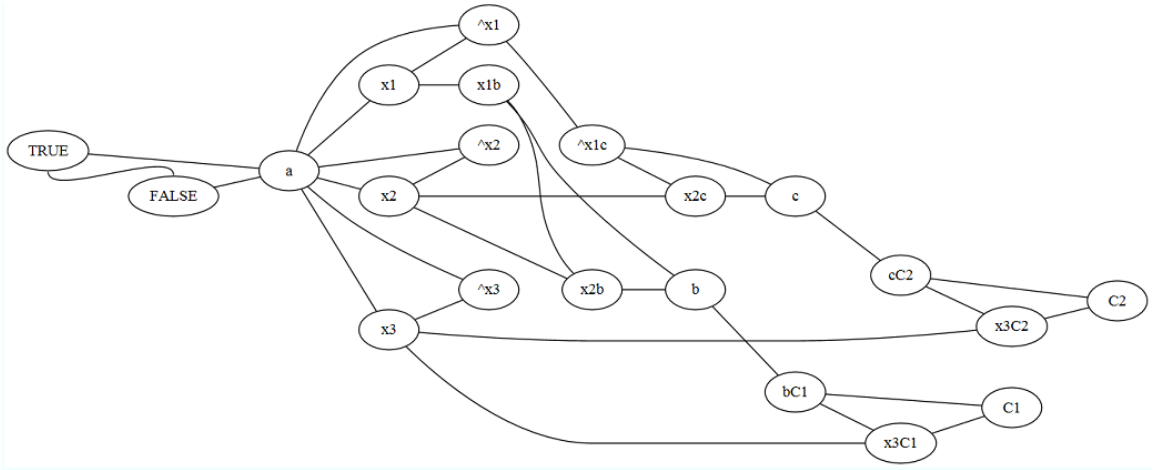
5.2 Původní řešení

Jak už bylo v kapitole 3.2 naznačeno, tak původně měl být informační systém klasická ASP.NET aplikace složená z komponent zvaných WebForms a jako hlavní implementační jazyk měl být použit C#. Mým prvním úkolem bylo naimplementovat převod 3-SAT problému na problém nezávislé množiny. Uživatelské rozhraní pro vstup 3-SAT problému se skládalo ze základních komponent technologie ASP.NET WebForms, jako jsou Label, TextBox, či DropDownList. Pro výstup, tedy problém nezávislé množiny, což je tzv. grafový problém, jsem ale potřeboval nějakou grafickou vizualizaci.

Zkoušel jsem vícero řešení, ale zmíním pouze jedno, nad kterým jsem strávil asi nejvíce času, a to knihovnu pod otevřenou licencí zvanou Graphviz (Graph Visualization Software) [10]. Hlavním problémem bylo, že jsem nebyl schopen zobrazit výstupní graf tak, jak jsem potřeboval a ani na uspokojivé grafické úrovni. Jediné, co graf zobrazoval správně, byla data (vrcholy a hrany), ale já

potřeboval zobrazovat graf mezi jednotlivými výpočty tak, aby z něj uživatel ihned pochopil, co se v něm (ve výstupu) při jednotlivých krocích změnilo. Prvky při každém kroku různě měnily své pozice, SVG výstup se na prohlížeči vykresloval při trochu vyšším počtu vrcholů a hran až příliš znatelně pomaleji atd. (jeden takový výstupní graf lze vidět na obrázku 20).

K tomuto zkrátka řešení přes C#, tedy serverově řízené programování, nebylo ideální. Zvláště tedy když se dalo počítat s tím, že značná část NP-úplných problémů, mezi nimiž jsem měl naimplementovat převody, budou tzv. grafové (nakonec jich bylo 5 z 8). Až po delším hledání jsem našel opravdu to ideální řešení.



Obrázek 20: Vizualizace grafu pomocí knihovny Graphviz

5.3 Knihovna vis.js

Pomalu tedy bylo jasné, že jsem zvolil špatnou cestu. K tomu, co bylo mým úkolem, jsem nutně nepotřeboval ani ASP.NET WebForms, ani nic podobného, nakonec ani samotný C#, jak jsem si původně dogmaticky myslel. Došlo mi, že se budu muset mnohem více zaměřit na klientskou část, ba dokonce že by mi možná mohla stačit pro celé řešení. Nakonec jsem po dalším průzkumu zvolil, že pro vykreslování grafů použiji JavaScritovou knihovnou vis.js [11].

Jedná se o dynamickou, na prohlížeči založenou knihovnu, která je navržena tak, aby byla snadně použitelná, zvládla velké množství dynamických dat a byla schopna manipulace a interakce s daty. Knihovna je pod licencí Apache 2.0 a MIT a může být šířena podle obou licencí. Skládá se z komponent DataSet, Timeline, Network, Graph2d a Graph3d. Já využil pouze komponenty DataSet a Network, které dále podrobněji popíšu.

5.3.1 Modul DataSet

Jedním z výchozích bodů vizualizace této knihovny je, že se dokáže vypořádat s dynamickými daty. Aby to bylo možné, tak knihovna zahrnuje flexibilní DataSet založený na dvojici klíč / hodnota a to nám umožňuje spravovat nestrukturovaná JSON data. To zjednodušeně řečeno znamená, že DataSet ukládá JSON objekty podle id. Tyto objekty lze nejen přidávat, aktualizovat, či mazat, ale také se umí přihlásit ke změnám v DataSetu. Data v DataSetu pak mohou být filtrována, seřazena apod.

Poskytnutím DataSetu s daty k vizualizacím, vizualizace může zjistit změny v DataSetu a tím pádem automaticky odrážet změny ve svém zobrazení. Změny v datech mohou být způsobeny serverem urgujícím aktualizace na klienta anebo také jinou interaktivní komponentou ve webové aplikaci. Obráceně mohou být zase data změněna v jedné z vizualizací. Například REST-klient může zjistit změny a tyto automaticky přepsat na serveru. Popřípadě mohou být poslány dalším částem webové aplikace.

REST Representational State Transfer je způsob pro čtení, editování či mazání informací ze serveru pomocí jednoduchých HTTP volání. Jedná se o distribuovanou architekturu, což znamená, že máme 2 části běžící na různých strojích, například naši webovou aplikaci kde webový prohlížeč komunikuje se serverem. REST umí jednotný, jednoduchý přístup ke zdrojům (resources). Ty mají svůj vlastní identifikátor URI a REST k nim přistupuje pomocí čtyř základních metod. REST je orientován datově, tudíž zdroji jsou nějaká konkrétní data. V našem případě, jak už bylo řečeno, slouží k popisu dat JSON (JavaScript Object Notation), jenž je odvozen z JavaScriptu. Jeho výhodami jsou nízká provozní režie a snadná a rychlá interpretace v jakémkoliv webovém prohlížeči. [12]

5.3.2 Modul Network

Tento modul (dále Network) je vizualizací zobrazení sítí, tedy sítí skládajících se z vrcholů a hran. Vizualizace je snadno ovladatelná a podporuje vlastní tvary, styly, barvy, velikosti, obrázky atd. Network vizualizace pracuje hladce na libovolném moderním prohlížeči až pro několik tisíc vrcholů a hran. Má podporu pro shlukování, aby zvládl zpracovat větší množství vrcholů. Pro vykreslování využívá HTML canvas. Od verze 4.0 se Network skládá z dalších modulů, které zpracovávají určité části sítě. Tyto moduly mají své vlastní dokumentace, možnosti, metody a události. Všechny tyto moduly alespoň ve zkratce popíšu, avšak ty které jsem hojně využil, popíši podrobněji.

Modul configure Generuje interaktivní volitelný editor s filtrováním, řeší samotnou HTML část canvasu.

Modul edges Možnosti pro hrany musí být obsaženy v objektu nazvaném edges. Všechny tyto možnosti mohou být nastaveny na každou hranu zvlášť. Z tohoto důvodu by měla mít každá hrana definované své vlastní id a nemělo by být definované globálně, nýbrž pro každou hranu zvlášť. Možnosti definované v globálním objektu edges jsou tedy aplikovány na všechny hrany, které nemají nastavené své vlastní, protože vlastní možnosti mají vyšší prioritu než ty globální. Pokud nastavíme hraně nějakou možnost, přepíšeme tu globální pro danou vlastnost. Na druhou stranu, pokud možnost nastavíme na *null*, tak se vlastnost nastaví na výchozí defaultní hodnotu.

Pro hrany jsem si vytvořil vlastní obecnou funkci, která do grafu (lépe řečeno DataSetu pro hrany) přidává hranu se všemi globálním vlastnostmi a v parametrech funkce jsou kromě daného DataSetu všechny vlastnosti, které má každá hrana své vlastní. Stěžejními parametry jsou vlastnosti $from$ a *to*, což jsou vlastně id vrcholů, které hrana spojuje. Dalším parametrem je zakřivení, které je u některých grafů potřeba pro lepší estetiku a také typ a velikost tohoto zakřivení. Pokud tyto parametry nejsou potřeba, tak jsou nastaveny na *null* a hrana pak žádné zakřivení nemá. Tato funkce vrací pro novou hranu automaticky vygenerované id. Tento modul obsahuje další desítky nastavitelných vlastností plus další vnořené do nižších úrovní, avšak pro potřeby implementovaných grafových NP-úplných problémů jsem pro tyto vlastnosti převzal defaultní hodnoty a nastavil je jako globální v rámci celé aplikace. Jediná vlastnost, která ještě stojí za zmínku je booleovský vnořený atribut *arrows.to.enabled*, který nastavuji na *true* nebo $false$ podle toho, zda pracuji s orientovaným, či neorientovaným grafem.

Modul nodes Možnosti pro vrcholy patří do objektu zvaného nodes. Stejně jako u hran platí, že mohou být nastaveny jak globálně, tak pro každý vrchol zvlášť, s tím že id by měl mít každý vrchol unikátní. Celá logika pro tento modul je obdobná jako u modulu edges.

Obdobně jsem si i zde vytvořil funkci, která přidává vrcholy do DataSetu, jenž je v parametru funkce. Dalšími argumenty funkce jsou vlastnosti, které má každý vrchol jiné. Jedním z nich je id vrcholu, protože si jej na rozdíl od hran, kde je automaticky generované, sám určuji. To z důvodu, že id vrcholu obvykle potřebuji znát v různých krocích animací převodů. Dále si zde posílám název vrcholu, jeho souřadnice x a y a také atribut *group*, který je blíže popsán v dalším modulu. Pokud není atribut *group* nastaven, tak se nastaví na hodnotu undefined převzatou z globálního nastavení, stejně tak jako design vrcholů, velikost popisků, barvy a další desítky atributů, které jsou už nastaveny globálně pro všechny animace stejně. Funkce nevrací nic, ani id vrcholu, protože jej přijímá jako parametr.

Modul groups Do tohoto modulu si můžeme přidat libovolný počet skupin, které mohou obsahovat většinu vlastností nacházejících se v modulu nodes u kterých to dává smysl. Například pravděpodobně nebudeme chtít nastavit stejné id nebo souřadnice x a y pro celou skupinu vrcholů. Skupiny si nadefinujeme obdobně jako vrcholy, jen s tím rozdílem, že každé skupině přiřadíme de facto libovolný unikátní název. Všechny vlastnosti nadefinované v nějaké skupině pak budou aplikovány na všechny vrcholy, kterým nastavíme atribut *group* na název této skupiny. Pokud by tento název neexistoval, tak se hodnota bere jako undefined a klasicky se převezmou globální vlastnosti.

V projektu se nacházejí 4 skupiny: *redGroup*, *greenGroup*, *blueGroup* a *redFontGroup*. Jak už názvy napovídají, tak první tři jsou pro vrcholy různých barev (v několika atributech). Pokud vrchol nemá přiřazenou skupinu, tak se vyberou globální vlastnosti a vrchol bude mít barvu šedou. Poslední jmenovaná skupina má stejné vlastnosti jako globální, akorát popisek je červenou barvou. Více skupin v projektu nebylo potřeba, ale pokud bych pro nějaký převod chtěl například změnit velikost vrcholů, velikost jejich popisků anebo třeba aby vrcholy byly reprezentovány nějakými ikonkami, tak by mi stačilo si pro tento převod nadefinovat novou skupinu s vlastnostmi, které chci pozměnit a ostatní vlastnosti se mi podědí z globálně nadefinovaného modulu nodes.

Modul physics Tento modul se stará o simulaci fyziky, řeší pohyb vrcholů a hran do jejich finálních souřadnic a upravuje stabilizaci. Je zde klíčový atribut zvaný *stabilization*, který pokud je nastaven na *true*, tak je síť automaticky stabilizována na zatížení s použitím výchozího nastavení. To může být dále upraveno dle potřeby. Já jsem ale tento atribut nastavil na $false$, tudíž jsem zakázal stabilizaci pro všechny animace. V podstatě to znamená, že kdykoliv změním graf, tak rozložení vrcholů a hran bude přesně takové, jaké si nadefinuji a elementy se v canvasu nebudou samovolně pohybovat. Na ukázkových projektech tyto pseudonáhodné pohyby a animace vypadají pěkně, ale pro cíle tohoto projektu jsou naprosto kontraproduktivní. Uživatel potřebuje sledovat změny v grafech a mít dokonalý přehled, kde se každý vrchol nachází, místo aby při každém kroku animace měnil svou pozici.

Modul layout Působí jako kamera sledující canvas. Umí animace, přibližování a ostření. Můžeme zde nastavit systém, jak rozložit prvky pro počáteční a hierarchické pozice v canvasu. Pokud například umožníme hierarchické uspořádání, tak přepíšeme některé další možnosti. Jelikož jsem v aplikaci nastavil u modulu physics atribut *stabilization* na $false$ a pro všechny vrcholy nastavuji souřadnice x a y sám, tak tuto sekci nemělo smysl využívat.

Modul manipulation Tento modul nám poskytuje API a volitelné GUI pro zadání změn dat v síti. Je zde klíčový atribut *enabled*, jehož výchozí hodnota je nastavena na $false$. Tento

atribut je v aplikaci většinou nastaven na $false$, až na dva případy. A to u prvního kroku dvou převodů - problému HC na problém HK a převodu problému HK na TSP, kdy je tento atribut nastaven na $true$. V této fázi má totiž uživatel možnost vytvářet či editovat vstupní graf.

Jsou zde naimplementovány metody pro přidání vrcholu, přidání hrany, editaci vrcholu, editaci hrany, smazání vrcholu a smazání hrany. Tyto metody lze v případě potřeby přepsat, či poupravit. Já si upravil metody pro přidání vrcholu, editaci vrcholu a přidání hrany. Ve funkci pro přidání vrcholu nastavuji jazyk (text) ve vyskakovacím okně dle potřeby a hodnotu pro atribut *label* (v okně je nazvaný $Name$) vytvářeného vrcholu, kde mám vlastní metodu generující poslední neobsazené číslo, aby uživatel nemusel pokaždé zbytečně zadávat název vrcholu. Funkce pro editaci vrcholu obsahuje jen jazykovou (textovou) úpravu vyskakovacího okna. U funkce pro přidání hrany nastavuji zakřivení hrany u editace vstupu problému HC, jelikož jsou zde orientované hrany. Pokud totiž vedou mezi dvěma vrcholy dvě hrany, každá s opačnou orientací, tak jsem chtěl, aby se míjely v oblouku a nepřekrývaly. U převodu, kde je vstupem problém HK, jsem zakřivení hrany nepotřeboval, jelikož zde nejsou orientované hrany (to tato funkce také rozlišuje). Je zde taky naimplementováno potvrzující okno vyskakující v momentě, kdy chce uživatel vložit hranu spojující vrchol sám se sebou.

Modul interaction Umí zpracovávat všechny interakce uživatele se sítí, např. kliknutí myší a dotekové události, stejně tak navigační tlačítka nebo vyskakovací okna. Této sekce jsem v aplikaci nevyužil, jelikož jsem nepožadoval přímou akci mezi uživatelem a sítí. Síť je totiž generována podle vstupu a algoritmu daného převodu.

5.4 Zadání vstupu

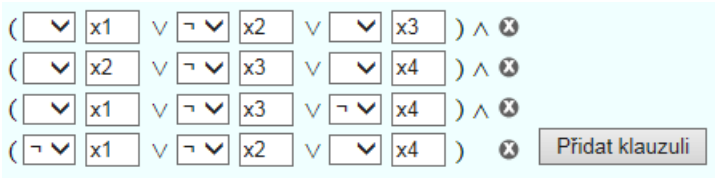
Ještě než začne samotná simulace převodu je potřeba zadat vstupní problém. Zadání vstupu se provádí v nultém kroku a to tak, že si uživatel může vybrat vstup z předem nadefinovaných příkladů, či si může vstup vytvořit sám.

5.4.1 Vlastní vstup

V aplikaci se nacházejí tři různé vstupní problémy, jejichž vlastní zadání (GUI a chování) se liší.

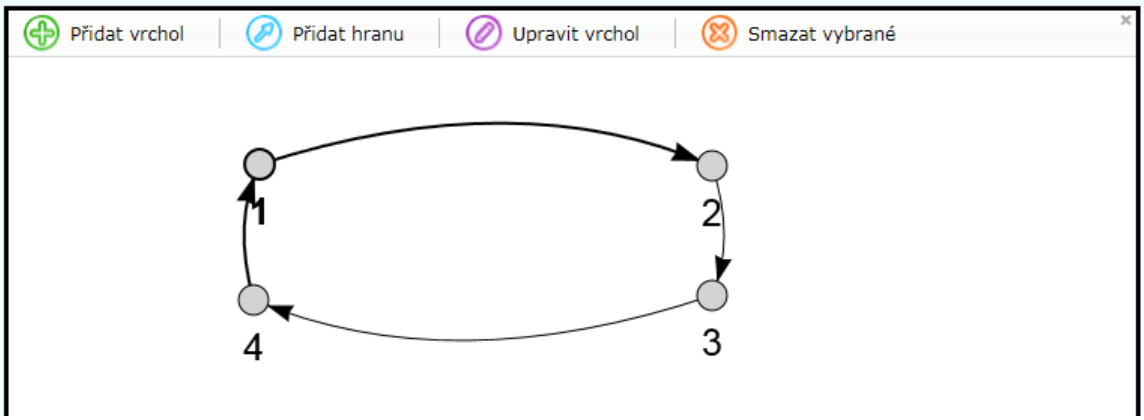
Vstup 3-SAT U zadání 3-SAT formule jsem využil toho, že každá klauzule musí obsahovat přesně 3 literály. Díky tomu jsem mohl vytvořit tabulku, kde každý řádek představuje jednu klauzuli při zachování toho, aby každý řádek vypadal stejně. Řádek obsahuje počáteční závorku, v níž jsou 3 literály oddělené znakem pro disjunkci. Každý řádek je zakončen znakem

pro konjunkci (kromě posledního) a tlačítkem s ikonkou křížku pro smazání příslušného řádku. Pro zadání každého literálu slouží dvojice složená z dropdownlistu, v němž jsou na výběr dvě možnosti, a to prázdné pole, či znak pro negaci a pak z textového pole určeného pro název proměnné. Toto textové políčko je z designových důvodů shora omezeno na 3 znaky s tím, že musí obsahovat alespoň 1 znak. Tato konstrukce mi zaručuje korektní zadání 3-SAT formule uživatelem. Celá tato komponenta ještě obsahuje tlačítko pro přidání klauzule s nevyplněnými literály na pozici posledního řádku v tabulce.



Obrázek 21: Komponenta pro zadání vstupu 3-SAT

Vstup HC Zadání hamiltonovského cyklu je už o něčem jiném, jelikož se jedná o graf. Bylo tudíž potřeba vytvořit rozhraní pro vytváření a editaci grafu. Při nultém kroku animace se u vstupního grafu v levém horním rohu nachází tlačítko *Upravit*, které otevírá možnosti pro editaci grafu. Metody a možnosti této naimplementované funkcionality jsou již blíže popsané výše, viz odstavec Modul manipulation, a tak nebudu zbytečně rozepisovat metody a možnosti této části. Ovládání a interakce s uživatelem je v této části velmi intuitivní.



Obrázek 22: Komponenta pro zadání vstupu HC

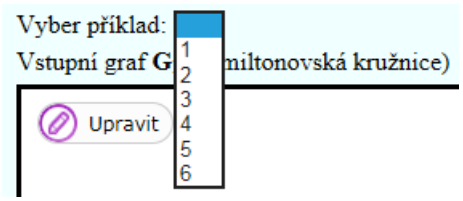
Vstup HK Zadání hamiltonovské kružnice je obdobné jako zadání hamiltonovského cyklu, jen s tím rozdílem, že při vkládání hran nezáleží na tom, kde začneme vkládání hrany a kam bude směřovat, protože hrany jsou zde neorientované.

Obrázek 23: Vyskakovací okno pro přidání vrcholu

5.4.2 Nadefinovaný vstup

Jednou ze zásad pro vypracování této práce bylo, aby převody obsahovaly i ukázkové vstupy a šlo tak zobrazit postup výpočtu bez nutnosti zadávání vstupu uživatelem. Uživatel má tedy možnost zvolit si z dropdownlistu již nadefinovaný vstup. Při prvním načtení stránky není v této nabídce vybraný žádný a uživatel si musí nějaký vybrat, pokud nechce vstup zadávat sám. Rozšířením tohoto požadavku je možnost pro poučenou osobu (například přečtením této kapitoly) upravit či přidat tyto nadefinované vstupy pro výběr uživatelem. K tomu slouží konfigurační soubory nacházející se ve zdrojových souborech ve složce *ConfigFiles/Examples*, kde jsou 3 soubory, každý pro jiný vstupní problém zvlášť.

U těchto souborů se předpokládá, že je může modifikovat i jiná osoba než programátor, například vedoucí této práce, či jiný pedagog. Jedinou podmínkou je, že u toho musí dotyčný dodržet pár pravidel. Společnou podmínkou pro všechny 3 soubory je, aby byl název funkce obsahující vstup složen ze slova *example* a čísla, které ještě nebylo použito, například *example4*. Pod tímto číslem se pak bude příklad zobrazovat v nabídce pro výběr vstupu. Další pravidla se liší podle konkrétního vstupního problému. Je třeba říct, že toto nadefinování se může zdát na první pohled komplikované, ale je třeba brát to jako nadstavbu, která nebyla požadována a do budoucna nemusí být ani využita a tudíž není nutno tyto soubory modifikovat.



Obrázek 24: Nabídka pro výběr ukázkového vstupu

Vstup 3-SAT Pro 3-SAT formuli je nutné pro každý vstup nadefinovat hodnoty literálů, kde jako znak pro negaci může být použit pouze znak „¬"a každý literál musí mít určenou pozici ve formuli následujícím způsobem:

```
function example3() {

   clause1Literal1 = 'x1';
   clause1Literal2 = 'x2';
   clause1Literal3 = '¬x3';

   clause2Literal1 = '¬x1';
   clause2Literal2 = 'x2';
   clause2Literal3 = '¬x3';
}
```

Výpis 1: Funkce pro přidání ukázkového vstupu 3-SAT

Tato konstrukce nám po výběru možnosti 3 (protože znak 3 je za klíčovým slovem *example*) z dropdownlistu pro výběr příkladu, zajistí správné nastavení komponenty pro 3-SAT formuli popsanou v podkapitole pro vlastní vstup, viz odstavec Vstup 3-SAT.

Vstup HC Nadefinování příkladu pro hamiltonovský cyklus je o něco složitější:

```
function example4(nodes, edges) {

   addNode(nodes, '1', '1', -200, 100, null);
   addNode(nodes, '2', '2', -100, -120, null);
   addNode(nodes, '3', '3', 200, 100, null);

   addEdge(edges, '1', '2', false, null, null);
   addEdge(edges, '1', '3', false, null, null);
   addEdge(edges, '3', '2', true, 'curvedCW', 0.2);
}
```

Výpis 2: Funkce pro přidání ukázkového vstupu HC

Argumenty funkce nodes a edges je potřeba zachovat, protože se zde posílají názvy proměnných, které program potřebuje znát. Parametry pro vložení vrcholu jsou pak id (musí být jedinečné), název, souřadnice x a y v canvasu a poslední argument zde není potřeba (je vždy *null*). Pro vložení hrany jsou pak důležité argumenty pro id počátečního a id koncového vrcholu (obě musí existovat), které hrana spojuje, poté booleovská hodnota zda chceme zakřivení hrany, popřípadě typ a velikost tohoto zakřivení.

Vstup HK Zadání hamiltonovské kružnice je opět obdobné jako zadání hamiltonovského cyklu, akorát zde u přidávání hran nezáleží na tom, který vrchol je počáteční a který koncový, jelikož hrany jsou zde neorientované. Ze stejného důvodu nepotřebujeme žádné zakřivení hran.

5.5 Zobrazení průběhu převodu

Tato kapitola nepopisuje teorii pro algoritmy převodů, ta je obsažena v kapitole 2.3. Zaměřuje se spíše na popis toho, jak jsou tyto algoritmy prezentovány uživateli a z menší části také na to, jak jsou tyto algoritmy implementovány.

V momentě kdy uživatel zadá vstup a klikne na tlačítko Další, tak se vstup uzamkne do stavu, kdy je pouze pro čtení a už jej nelze modifikovat. Na jeho modifikaci se dá vrátit pouze po kliknutí na tlačítko Předchozí u prvního kroku převodu. V této chvíli se vypočítá celkový počet kroků celé animace daného převodu a aktuální krok je nastaven na 1. Informace v labelu pro aktuální krok vypadá pak např. takto: 1/35.

5.5.1 Převod 3-SAT na IS

Poznámka: Teoretický popis tohoto algoritmu je popsán v kapitole 2.3.1.

První kroky převodu jsou věnovány základnímu popisu převodu a pokračuje se popisem sestrojení nezávislé množiny. V jednotlivých krocích jsou nejdříve vytvořeny základní trojúhelníky za každou klauzuli a pak se v cyklu za každou proměnnou vkládají další hrany s odpovídajícím popisem, kdy je pro každou proměnnou vyhrazen 1 krok.

Po sestrojení grafu je uživateli nabídnuto, aby ohodnotil vstupní proměnné tak, aby vstupní 3-SAT formule byla splnitelná. Poté mu je prezentováno, nalezení nezávislé množiny zvýrazněním jednoho pravdivého literálu z každé klauzule jak ve vstupní formuli, tak ve výstupním grafu. Tím mu je předvedeno, že vstupní formule je splnitelná a zároveň výstupní graf obsahuje nezávislou množinu.

V poslední fázi mu je prezentován opačný případ, tedy nejdříve nalezená nezávislá množina a poté pravidla pro ohodnocení vstupních proměnných dle nalezené nezávislé množiny. Na základě těchto pravidel jsou pak uživateli vypsány všechny kombinace možných pravdivých ohodnocení formule pro tuto nezávislou množinu.

5.5.2 Převod 3-SAT na 3-CG

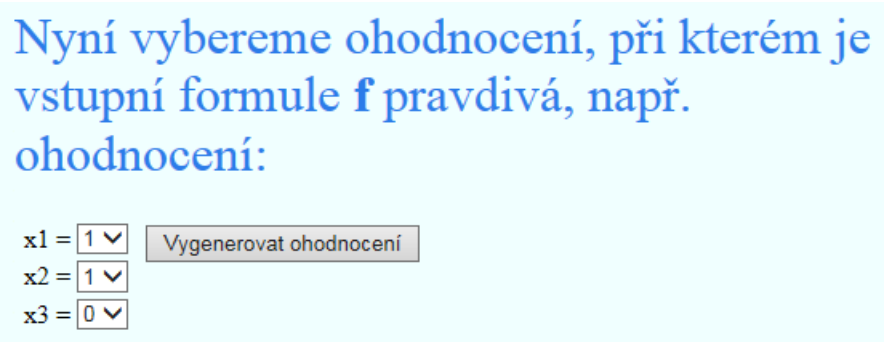
Poznámka: Teoretický popis tohoto algoritmu je popsán v kapitole 2.3.2.

Na začátku algoritmu se vypočítají rozestupy v ose y mezi jednotlivými vrcholy, aby se zajistilo, že se všechny potřebné vrcholy vmístí na plátno. Tyto rozestupy se spočítají dle toho, kolik klauzulí a proměnných vstupní 3-SAT formule obsahuje, jelikož tyto faktory pak určují podobu a velikost výstupního grafu.

Poté se přejde k počátečním krokům a základnímu popisu převodu. Postupně se začne vytvářet výstupní graf. Nejdříve vytvoření trojúhelníku složeného ze tří základních vrcholů pro každou barvu, na to navazují vrcholy pro všechny proměnné a jejich negace s příslušnými hranami. V dalších krocích je uživateli na první klauzuli krok za krokem vysvětlena konstrukce, která se vytváří pro klauzule. V této fázi se také průběžně prezentují kombinace a principy barvení vrcholů. Až je vše vysvětleno, tak se v cyklu pro každou klauzuli vytvoří již zmiňované konstrukce.

Jakmile je graf sestrojen, tak je uživateli nabídnuto, aby ohodnotil všechny proměnné vstupní formule tak, aby byla formule při daném ohodnocení pravdivá. Na základě tohoto ohodnocení se krok po kroku obarvuje výstupní graf s vysvětlujícími komentáři, proč zrovna jaká barva. Fáze je ukončena tím, že celý graf je korektně obarven třemi barvami, jelikož vstupní formule byla splnitelná.

Ve finální fázi uživatel vyplňuje opačné ohodnocení, tzn. takové, při kterém nebude vstupní formule pravdivá. Následuje to stejné co v předešlém cyklu, tedy barvení grafu s popisem. Alespoň do té doby, než se narazí na konflikt v barvení grafu, tedy do momentu kdy 2 sousední vrcholy budou mít nevyhnutelně stejnou (červenou) barvu. Uživateli je tedy znázorněno, že při tomto ohodnocení vstupních proměnných neobarví graf správně.



Obrázek 25: Nabídka pro ohodnocení proměnných

5.5.3 Převod 3-SAT na SUBSET-SUM

Poznámka: Teoretický popis tohoto algoritmu je popsán v kapitole 2.3.3.

V úvodním kroku je vytvořená jednoduchá HTML tabulka reprezentující výstupní instanci problému SUBSET-SUM. V dalším je tato tabulka naplněna hodnotami a jsou popsány jednotlivé řádky a sloupce v ní. Následuje cyklus, kde je podrobně popsán každý řádek určený pro literál vstupní 3-SAT formule. Tento cyklus lze přeskočit tlačítkem *Přeskočit*. Následuje obdobný cyklus, který lze rovněž přeskočit, avšak tentokrát s popisem sloupců odpovídajícím jednotlivým proměnným vstupní formule. Poslední cyklus pak popisuje sloupce, které odpovídají klauzulím vstupní formule.

Jakmile je vysvětlena konstrukce celé tabulky, tak je uživatel vyzván k takovému ohodnocení proměnných vstupní formule, aby formule byla pravdivá. Poté mu je vysvětlen postup pro výběr řádků reprezentující čísla z množiny (jaké množiny je popsáno v teoretické kapitole o tomto převodu) z tabulky a v navazujícím kroku jsou tyto řádky vybrány, respektive opačné jsou smazány. U sloupců, co nám zbyly, jsou pak ještě zelenou barvou zvýrazněna čísla, která dávají součet z posledního řádku (ten je také obarven). Je tak přehledně vidět výstupní množina i suma problému SUBSET-SUM.

	x1	x2	x3	x4	c1	c2	c3
z1	1	0	0	0	0	0	1
y2	0	1	0	0	0	1	1
y3	0	0	1	0	1	1	0
z4	0	0	0	1	0	0	0
g1	0	0	0	0	1	0	0
h1	0	0	0	0	1	0	0
g2	0	0	0	0	0	1	0
g3	0	0	0	0	0	0	1
t	1	1	1	1	3	3	3

Obrázek 26: Podmnožina s odpovědí ANO

Poslední fáze simulace převodu je obdobná jen s tím rozdílem, že uživatel zadává takové ohodnocení proměnných, při kterém není vstupní formule pravdivá. Opět mu jsou vysvětlena pravidla pro výběr řádků a jsou obarvena čísla dávající součty ve sloupcích i součty samotné. V tomto případě však nejsou všechny hodnoty obarveny zelenou barvou, jelikož v některých sloupcích

nám součty nesedí. Pro tyto případy nám poslouží červená barva. Uživateli je ještě na závěr animace vysvětleno, proč tomu tak je, ve zkratce tedy že vstupní formule není pravdivá a tudíž ani nelze nalézt množinu takovou, aby dávala požadovanou sumu.

	x1	x2	x3	x4	c1	c2	c3
y1	1	0	0	0	1	1	0
z2	0	1	0	0	1	0	0
y3	0	0	1	0	1	1	0
z4	0	0	0	1	0	0	0
g2	0	0	0	0	0	1	0
g3	0	0	0	0	0	0	1
h3	0	0	0	0	0	0	1
t	1	1	1	1	3	3	3

Obrázek 27: Podmnožina s odpovědí NE

5.5.4 Převod 3-SAT na ILP

Poznámka: Teoretický popis tohoto algoritmu je popsán v kapitole 2.3.4.

V počátečních krocích převodu jsou vysvětleny základní definice, přiřazení náhradních proměnných nerovnicím a jsou představeny základní nerovnice. Následuje sekce, kde jsou krok za krokem prezentovány nerovnice pro každou klauzuli, kterou lze přeskočit.

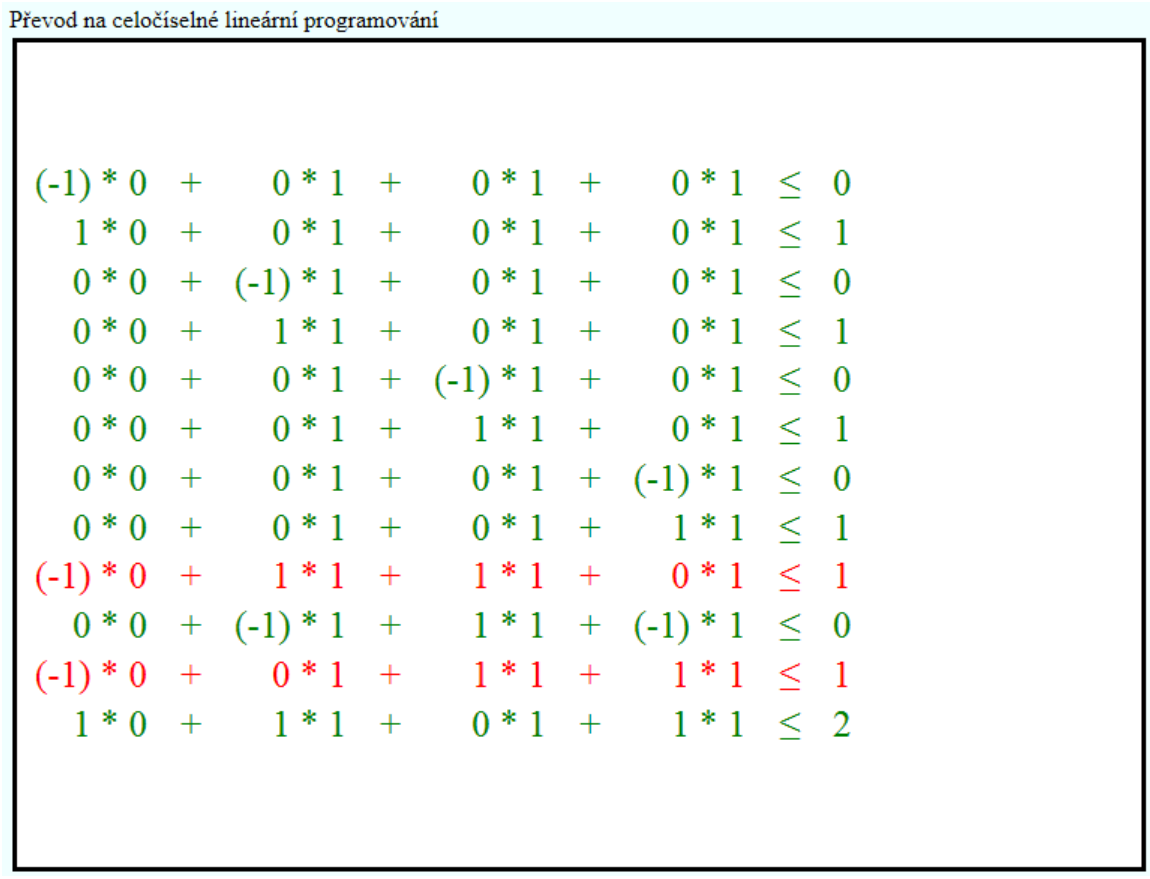
Jakmile jsou všechny nerovnice známy, je uživateli vysvětleno, do jakého stavu je potřeba nerovnice převést. Poté začne sekce, kde je v jednotlivých krocích vyobrazena úprava každé z nerovnic určených pro klauzule. Všechny aritmetické úpravy jsou zde uživateli náležitě vysvětleny. Tuto sekci lze také přeskočit, což se hodí třeba v momentě, kdy máme 8 klauzulí, a aritmetické úpravy nerovnic uživatele nezajímají.

V okamžiku kdy jsou všechny nerovnice upraveny, je uživateli zobrazena celá soustava nerovnic a v dalším kroku i v maticovém zápisu tak, jak si ji problém lineárního programování vyžaduje.

Obě tato zobrazení jsou zprostředkována pomocí dynamicky vytvořených tabulek, a tudíž jsou odpovídající proměnné setříděny přehledně pod sebou. Následně je uživateli v několika krocích vysvětleno nadcházející dosazování hodnot do nerovnic za proměnné.

Potom už klasicky následuje výzva, aby uživatel vyplnil, popřípadě vygeneroval ohodnocení proměnných vstupní 3-SAT formule tak, aby formule byla pravdivá. Díky tomu můžeme dle pravidel, která byla vysvětlena v předchozích krocích převodu, dosadit hodnoty do nerovnic a vyhodnotit zda všechny nerovnice platí. Nerovnice jsou mimochodem opět vsazeny do dynamicky vytvořené HTML tabulky, kde je každá nerovnice na samostatném řádku a proměnné ve sloupcích pod sebou tak, aby vjem uživatele byl co nejlepší. Nerovnice, které platí, jsou v tomto kroku obarveny zeleně. A to jsou vlastně všechny, protože jak víme, vstupní formule byla pravdivá, a tudíž musí platit všechny nerovnice (odpověď *ANO* musí být pro oba problémy zachována).

Převod je jako obvykle zakončen také tím, že uživatel zadá či si vygeneruje ohodnocení proměnných, pro které není vstupní 3-SAT formule pravdivá. Opět se dosadí příslušné hodnoty a v posledním kroku jsou opět vyhodnoceny všechny nerovnice, avšak zde nejsou všechny zelené (platné), ale nově je alespoň 1 červená, tedy neplatná. Uživatel tak vidí, že odpověď *NE* byla pro oba problémy také zachována.



Obrázek 28: Vyhodnocení s neplatnými nerovnicemi

5.5.5 Převod HC na HK

Poznámka: Teoretický popis tohoto algoritmu je popsán v kapitole 2.3.5.

Po úvodním kroku s krátkou definicí převodu jsou vytvořeny všechny vrcholy ve výstupním grafu pro hamiltonovskou kružnici, samozřejmě s patřičným odůvodněním. Následuje pasáž, kdy jsou za každou hranu vstupního grafu přidávány hrany do výstupního grafu. Za každou hranu je 1 krok s komentářem, kdy se postupně formuje výstupní graf. Proto je zde opět volba *Přeskočit*, jelikož nemá smysl vidět de facto to stejné třeba 15x za 15 hran.

Jakmile je výstupní graf sestrojen, vyhodnotí se algoritmem, zda vstupní graf obsahuje hamiltonovský cyklus, či ne. Pokud by jej neobsahoval, tak se algoritmus v dalším kroku pokusí ve výstupním grafu vyhledat hamiltonovskou kružnici, ale samozřejmě ji nenajde, jelikož musí být zachována odpověď NE pro oba problémy zároveň. Uživatel je s tím seznámen v obou krocích i závěrečným vyhodnocením.

Naopak pokud by algoritmus pro vyhledání hamiltonovského cyklu našel, co hledá, tak by animace převodu pokračovala dále. Následovala by fáze, kde by za každou hranu ze vstupního grafu obsaženou v hamiltonovském cyklu byla krok po kroku nalezena ekvivalentní hrana ve výstupním grafu a obarvena červeně. Pokud by uživatel opět nevyužil možnosti *Přeskočit*, tak by se postupně obarvily hrany ve výstupním grafu tak, že na konci by červené hrany tvořily hamiltonovskou kružnici ve výstupním grafu. Toto je ostatně na konci této sekce ověřeno i algoritmem. Uživatel tak vidí, že pro oba problémy byla zachována odpověď ANO.

Algoritmus pro vyhledání hamiltonovského cyklu nebo kružnice

Matice sousednosti V předmětu Metody analýzy dat 1 (MAD 1), který jsem absolvoval, byla polovina výuky věnována datovým sítím. Ty jsou de facto reprezentovány vrcholy a hranami, což i grafy, ve kterých jsem potřeboval najít hamiltonovský cyklus či kružnici. Díky znalostem z této oblasti jsem tudíž věděl, že pro efektivní reprezentaci grafu mohu použít tzv. matici sousednosti.

Matice sousednosti je v podstatě dvourozměrné pole, kde každý řádek i sloupec reprezentuje vrchol v grafu. Hodnoty v konkrétních buňkách pak reprezentují hrany mezi těmito vrcholy. Pokud je například v buňce na 3. řádku a ve 4. sloupci hodnota 1, tak z třetího vrcholu vede hrana do čtvrtého vrcholu. Pokud takováto hrana v grafu neexistuje, tak je zde hodnota 0. Za připomínku také stojí, že pokud je graf neorientovaný (například při hledání hamiltonovské kružnice), tak v buňce na 4. řádku ve 3. sloupci bude také hodnota 1, protože cesta vede i v opačném směru (neorientovaná hrana).

Funkce, kterou jsem na sestavení takovéto matice vytvořil je potom poměrně jednoduchá. Jejími argumenty jsou DataSet pro vrcholy, DataSet pro hrany, pole s id vrcholů a příznak, zda jsou hrany orientované, či ne. Nejprve si vytvořím dvourozměrné pole, kde počet řádků i sloupců je stejný jako počet vrcholů v DataSetu a každá hodnota v poli je nastavena na 0. Poté v cyklu procházím celý DataSet hran, kde každá hrana obsahuje id vrcholu, ze kterého vede (dále jako hodnota $from$) a id vrcholu, do kterého vede (dále jako hodnota to). Při tomto průchodu tedy do pole na řádek v matici odpovídající indexu hodnoty $from$ v poli id vrcholů a do sloupce odpovídajícímu zase indexu hodnoty to v tomto poli, uložím hodnotu 1. Pokud je příznak nastaven na $false$ (graf je neorientovaný), tak je obdobně nastavená hodnota 1 i na pozici s prohozeným indexem řádku a sloupce (cesta vede oběma směry). Až cyklus skončí, tak funkce vrátí hotovou matici sousednosti reprezentující graf, který je charakterizován výše zmíněnými vstupními parametry.

Hledání cesty v grafu A teď k samotnému hledání uzavřené cesty v grafu. Hlavní funkce nazvaná $hamiltonianCycleOrPath$ pro hledání cesty na začátku vytvoří pole (dále $path$) reprezentující případnou cestu o velikosti počtu vrcholů a každou hodnotu nastaví na defaultní hodnotu -1. Pak nastaví automaticky na nultou pozici hodnotu id prvního vrcholu, protože nezáleží na tom, kde začneme, ať už v cyklu, či kružnici – stejně vždycky musíme projít všechny vrcholy a pak se vrátit na začátek.

Poté se zavolá rekurzivní funkce $hamiltonianCycleOrPathUtil$, jejímž hlavním parametrem je aktuální hledaná pozice, což je při prvním zavolání 1, protože nultou pozici jsme už určili na začátku. Hned na začátku této funkce se ověří, zda aktuální pozice se rovná počtu vrcholů v grafu a popřípadě se také ověří, zda vede hrana z předposlední pozice do první, tedy jestli je cesta uzavřená. Pokud ano, tak funkce vrátí $true$ a rekurze je ukončena, protože jsme našli uzavřenou cestu. Pokud ne, tak se v cyklu projdou všechny vrcholy, zda mohou být přidány do cesty.

Na prověření, zda vrchol může být přidán do pole $path$, slouží další pomocná funkce $isSafe$, která ověřuje, zda hledaný vrchol má hranu s posledním vrcholem uloženým v poli $path$. Také ověřuje, zda tento vrchol už není v poli $path$. Pokud ověření funkcí $isSafe$ projde, tak se vrchol přidá do pole $path$ a rekurzivně se opět zavolá funkce $hamiltonianCycleOrPathUtil$ s argumentem pozice o 1 vyšší.

Toto se děje pořád dokola, dokud existuje nějaký neprověřený vrchol. Až se prověří všechny vrcholy a nenajde se hrana na přidání, tak se hodnota na aktuální pozici v poli $path$ nastaví na naši defaultní hodnotu -1 a rekurze se posouvá zpátky. Celý algoritmus tedy končí buď nalezením první cesty, kterou najde (jak je řečeno výše - toto se ověřuje hned na začátku této rekurzivní funkce) anebo poslední („nejvyšší“) rekurze vrátí $false$ a tím pádem víme, že se uzavřená cesta v grafu nenachází.

5.5.6 Převod HK na TSP

Poznámka: Teoretický popis tohoto algoritmu je popsán v kapitole 2.3.6.

Prvním krokem je klasicky definice převodu a poté se ihned přejde k sestrojování výstupního grafu a to ve dvou krocích. V prvním z nich se do výstupního grafu vloží pouze hrany, které jsou již ve vstupním. Tyto hrany jsou obarveny černou barvou a do legendy je vložena černým fontem poznámka o tom, že délka těchto hran je 1. V tom dalším se do výstupního grafu vloží hrany tak, aby výstupní graf byl úplný, tzn., aby mezi každou dvojicí vrcholů ve výstupním grafu byla hrana. Dá se to chápat tak, že se výstupní graf jakoby doplní. Tyto hrany jsou pak obarveny zelenou barvou stejně tak jako legenda říkající, že mají délku 2.

Tímto je výstupní graf sestrojen a přejde se k další fázi. Algoritmus se pokusí ve vstupním grafu nalézt hamiltonovskou kružnici. Pokud ji nenajde, tak se sice v dalším kroku pokusí vyhledat i okružní jízdu délky L (počet vrcholů v obou grafech) ve výstupním grafu, ale tu také samozřejmě nenajde. Uživatel je o tom informován spolu s patřičným závěrem.

Pokud algoritmus ve vstupním grafu najde hamiltonovskou kružnici, tak začne v dalších krocích postupně barvit červenou barvou ty hrany z výstupního grafu, které odpovídají těm z hamiltonovské kružnice vstupního grafu. Toto barvení probíhá tak, že se v 1 kroku obarví 1 hrana a tak lze tuto pasáž přeskočit. V tomto případě tedy uživatel na konci vidí ve výstupním grafu okružní jízdu požadované délky L a je mu předložen závěr, který to potvrzuje a příslušně vyhodnotí.

5.6 Texty v popisech převodů

Ze začátku byly popisy jednotlivých kroků všech převodů, tlačítek, labelů apod. zapsány v programu a na stránkách tzv. natvrdo. Časem ale vznikla potřeba mít tyto texty sjednocené a snadno editovatelné, aby nebylo nutné např. kvůli změně jednoho slova či věty prohledávat celý program. Za prvé by to bylo nepohodlné a za druhé zde hrozilo zbytečné riziko pro vznik potenciálních chyb. Dalším důvodem bylo, aby bylo jednoduché změnit jazyk např. z češtiny na angličtinu pro zahraniční studenty.

Zavedl jsem tedy pro každý převod zvlášť konfigurační soubor, kde se nastavují všechny texty vyskytující se zde. Vzniklo i pár dalších pomocných souborů tak, aby byl snadno editovatelný všechen text nacházející se v aplikaci. Tyto soubory se nacházejí ve složce *ConfigFiles/Text*. Slouží např. k tomu, abych já anebo v budoucnu vedoucí práce či jiná pověřená osoba mohl rychle a jednoduše editovat například popis jednotlivých kroků převodu. Jsou zde 2 druhy komentářů.

První typ je jednoduchá proměnná, které je přiřazený nějaký text v uvozovkách a to vše je zakončeno středníkem. Tato proměnná má vždy název takový (anglicky), aby se z ní dalo vyčíst, kde se v animaci zhruba nachází a co popisuje. Navíc se to dá poznat i z tohoto textu jako takového.

Dalším typem je jednoduchá funkce v JavaScriptu, která opět vrací řetězec. Funkci bylo třeba použít, aby bylo možno zakomponovat do řetězce komentáře, u kterých se dalo očekávat, že zde budou použity (což ale samozřejmě nemusí).

Soubory odevzdané v rámci této diplomové práce jsou zcela validní a nemusí být měněny. Pokud ale pověřená osoba bude znát základní znalosti syntaxe JavaScriptu, jako například skládání řetězce, pochopit rozdíl mezi proměnnou reprezentující řetězec a funkcí vracející řetězec a nebude měnit názvy těchto proměnných a funkcí, tak může měnit hodnoty těchto řetězců v těchto souborech. Po uložení aktualizovaných souborů na server, kde jsou tyto soubory nahrané, se změní i texty na stránkách tohoto serveru, protože se budou načítat nové hodnoty.

Dalším důvodem proč by někdo mohl cítit potřebu měnit obsah těchto textů, by mohla být změna stylů. Sám jsem tyto texty obohatil o nejrůznější styly (barvy, tučně, kurzíva, dolní indexy atd.) podle potřeby, kdy to přišlo vhod tak, aby uživatel mohl co nejjednodušeji a nejrychleji pochopit, co komentář říká. Toto stylování mi zabralo poměrně hodně času, ale myslím si, že jeho efekt na uživatelův vjem je značný a jelikož jedním z hlavních cílů této aplikace má obecně být vysvětlení a pochopení dané látky, tak je velice na místě.

Tento styl je proveden pomoci jednoduchých HTML tagů k těmto stylům určeným. Řetězce jsou totiž v programu skrze tyto proměnné a funkce vsázeny přímo do HTML obsahu stránek. Pokud tedy někdo bude chtít přidávat či měnit již zavedené styly, tak musí ovládat alespoň základy HTML, tzn., měl by vědět, jak funguje párový tag a nějaké konkrétní tagy pro změnu fontu znát (je jich jen pár).

Všechny HTML stránky v aplikaci obsahují odkazy na JavaScript a CSS souborech. Každá takováto reference pak obsahuje verzi, která je pro všechny tyto soubory jednotná. Pokud jsou některé soubory změněny, jako například ty pro text, tak doporučuji aktualizovat i tyto verze na novější hodnoty, aby se uživatelům při najetí na stránku nahrály na klienty tyto soubory znovu. Pokud by totiž měli nahrané staré verze těchto souborů, tak by se jim pořád mohly zobrazovat staré texty, minimálně do aktualizace stránky (klávesnice F5). Doporučuji pro lepší přehlednost změnit tyto verze všude stejně na větší a opět jednotnou hodnotu. Ve výpisu kódu 3 pak můžeme vidět příklad starší verze s časovou značkou 22.03.2016 12:00:00, která může být aktualizována na verzi s časovou značkou 13.04.2016 12:00:00, viz výpis kódu 4.

```
<script type="text/javascript" src="ConfigFiles/Text/Text_3SAT-IS.js?v
    =20160322120000"></script>
```

Výpis 3: Příklad reference na starší původní verzi

```
<script type="text/javascript" src="ConfigFiles/Text/Text_3SAT-IS.js?v
    =20160413120000"></script>
```

Výpis 4: Příklad reference na novější aktualizovanou verzi

5.7 Přidání dalších převodů

Pokud by chtěl někdo v budoucnu rozšířit stávající aplikaci o další převody, tak může jednoduše přidat další HTML stránku do projektové solution, popřípadě může využít technologie ASP.NET a jejich komponent a přidat entitu zvanou Web Form. Na místě je také rozšířit stávající menu o novou položku.

Jestliže by se převod skládal z problémů, které nejsou v aplikaci naimplementované, tak je samozřejmě potřeba je doimplementovat. Pro grafové problémy je však možno využít knihovnu vis.js popsanou v kapitole 5.3. Pokud by převod obsahoval problém, který je již naimplementovaný, popřípadě by se hodilo využít funkcionality, která se v aplikaci již nachází, tak je možno využít některé užitečné zdrojové soubory ze složky *Reductions/Utilities*, detailněji popsané níže:

- *3SAT-Evaluation.js* - obsahuje potřebné funkce pro ohodnocení proměnných 3-SAT formule a pro práci s tímto ohodnocením
- *3SAT-Formula.js* - nachází se zde nezbytné funkce pro práci s komponentou určenou pro vstupní formuli problému 3-SAT
- *Network-Edit.js* - jsou zde funkce potřebné pro komponentu přes kterou je možno vytvořit či editovat graf
- *Network-Hamiltonian.js* - nachází se zde funkce pro vytvoření matice sousednosti reprezentující graf a funkce pro vyhledání hamiltonovského cyklu či hamiltonovské kružnice v grafu na základě této matice
- *Network-Options.j*s - obsahuje konfiguraci vlastností všech grafů nacházejících se v aplikaci
- *Page.js* - jsou zde funkce pro posouvání a následné zobrazení aktuálního kroku

Je potřeba dodat, že tyto soubory potřebují ke správné funkčnosti HTML elementy, se kterými pracují, jejichž id a vlastnosti se dají vyčíst z těchto souborů anebo ze samotných HTML kódů stránek příslušných převodů.

6 Nasazení

Existují dvě možnosti nasazení této aplikace. Pro původní možnost je nutný server podporující technologii ASP.NET. Pokud takový server máme, tak nám stačí zkopírovat soubory ze složky *Deploy* do příslušného adresáře webového serveru. Aplikace je poté plně funkční a není potřeba žádné dodatečné nastavování. Aplikace byla v průběhu vývoje nasazována u českého poskytovatele webhostingu aspone.cz, specializující se na podporu technologie ASP.NET. Hlavní výhodou tohoto poskytovatele je, že v základní verzi je zcela zdarma a bez reklam při zaručení mnohonásobně větší diskové kapacity, než je pro tuto aplikaci potřeba. Konkrétně byla aplikace průběžně nasazovaná na adresu http://npuplneproblemy.aspone.cz/Home.html, kde byla pravidelně testována a prezentována vedoucímu práce. Na tomto serveru nebylo zjištěno žádné odlišné chování od lokálního virtuálního serveru poskytnutého vývojovým prostředím Microsoft Visual Studio.

Jak už jsem zmínil v kapitole 3.2, tak nakonec není ani potřeba podpora technologie ASP.NET. Toto jsem také otestoval u poskytovatele bezplatného webhostingu webzdarma.cz, který nemá podporu této technologie. Opět zde nebylo zjištěno žádné odlišné chování, a tudíž aplikace může být nasazena i na klasickém webhostingu. Stačí k tomu opět pouze zdrojové soubory zkopírované ze složky *Deploy* do příslušného adresáře webového serveru. Doporučil bych však pro tuto volbu také webhosting bez reklam, jelikož neměly dobrý vliv na estetiku stránek.

Je tedy jedno, jaké možnosti bude pro nasazení aplikace využito, obě jsou zcela funkční. Původní možnost zůstala zachována pouze z důvodu lepšího potenciálního navázání na tuto práci například jinou diplomovou prací či semestrálním projektem rozšiřující tuto aplikaci. Byla by podle mě škoda zbytečně zrušit možnost využití technologie ASP.NET jenom z důvodu, že není potřeba, jelikož nečiní žádný problém ve funkčnosti stávající aplikace.

7 Závěr

Cílem této diplomové práce bylo vytvořit server, na kterém si studenti budou moci zobrazit průběh převodů mezi NP-úplnými problémy. Celkem aplikace obsahuje 6 převodů, u nichž si uživatel může po jednotlivých krocích postupně zobrazit jejich průběh. Pro každý převod je zde možnost vybrat si nadefinovaný ukázkový vstup, ale také zadat vstup vlastní a aplikace dle něj dokáže postupně zobrazit všechny kroky převodu s odpovídajícím vysvětlujícím popisem. Průběh a popis tohoto převodu tedy není pokaždé stejný, ale mění se dynamicky v závislosti na daném vstupu. Server také obsahuje sekci s nezbytnou teorií k danému tématu.

Celé řešení je koncipováno tak, aby bylo co nejjednodušší práci rozšířit o nové převody jak mezi implementovanými problémy, tak mezi novými. Projekt není ani nijak přísně vymezen technologiemi, jelikož lze rozšířit jak o klasické HTML stránky obohacené o kód např. v JavaScriptu, tak například o ASP.NET WebForms, jejichž chování by bylo řízené v jazyce C#, či jiném. Kladl jsem přísný důraz na hierarchii adresářů a zdrojových souborů tak, aby jednotlivé komponenty byly co nejjednodušeji znovupoužitelné. Aplikace má také solidní konfigurační možnosti, které nebyly v původním zadání požadovány, ať už jde o možnost editace textu v celé aplikaci, či například nastavení ukázkových vstupů pro jednotlivé převody.

Vidím v této práci potenciál pro kvantitativní rozšíření, jelikož existuje spousta dalších zajímavých NP-úplných problémů a tedy i převodů mezi nimi. Každopádně je ale tato práce jako samostatný celek ke svému účelu postačující a věřím, že najde po svém nasazení na server uplatnění a pomůže ve výuce této oblasti.

Literatura

[1] Prof. RNDr. Petr Jančar, CSc.: Teoretická informatika (učební text), VŠB-TUO FEI, 2007 (a 2010).

[2] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (STOC '71). ACM, New York, NY, USA, 151-158. DOI=http://dx.doi.org/10.1145/800157.805047

[3] Korespondenční Seminář z Programování. *Recepty z programátorské kuchařky, Těžké problémy* [online]. MFF UK. [cit. 2016-04-20]. Dostupné z: http://ksp.mff.cuni.cz/tasks/27/cook4.html

[4] Wikipedia: the free encyclopedia. *NP-úplnost* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2016-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/NP-%C3%BAplnost

[5] Wolfram MathWorld - The Web's Most Extensive Mathematics Resource. *Independent Set* [online]. Eric W. Weisstein [cit. 2016-04-20]. Dostupné z: http://mathworld.wolfram.com/IndependentSet.html

[6] Wikipedia: the free encyclopedia. *Graph coloring* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2016-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Graph_coloring

[7] Wolfram MathWorld - The Web's Most Extensive Mathematics Resource. *Hamiltonian Cycle* [online]. Eric W. Weisstein [cit. 2016-04-20]. Dostupné z: http://mathworld.wolfram.com/HamiltonianCycle.html

[8] Quora - The best answer to any question. *How to solve the traveling salesman problem using dynamic programming* [online]. Pankaj Kumar, 2013 [cit. 2016-04-20]. Dostupné z: https://goo.gl/kR2eCV

[9] *Reveal.js: The HTML Presentation Framework* [online]. Stockholm, Sweden: Hakim El Hattab, 2016 [cit. 2016-04-20]. Dostupné z: https://github.com/hakimel/reveal.js

[10] *Graphviz: Graph Visualization Software* [online]. Eclipse Public License - v 1.0, 2014 [cit. 2016-04-20]. Dostupné z: http://www.graphviz.org/

[11] *Vis.js: A dynamic, browser based visualization library* [online]. Almende B.V., 2016 [cit. 2016-04-20]. Dostupné z: http://visjs.org/

[12] Wikipedia: the free encyclopedia. *Representational State Transfer* [online]. San Francisco (CA): Wikimedia Foundation. [cit. 2016-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Representational_State_Transfer

Seznam příloh

A Příloha na CD

- **Deploy** - soubory potřebné k nasazení na server
- **DP_KUB0240** - aplikace a její zdrojové soubory
- **TeX** - zdrojové soubory pro LaTeX, včetně použitých obrázků
- **Text** - text diplomové práce